Agent-Based Mediated Service Negotiation

Mobach, D.G.A.

2007

document version
Publisher's PDF, also known as Version of record

Agent-Based Mediated Service Negotiation

David Mobach

SIKS Dissertation Series No. 2007-09

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems, and has been financially supported by Stichting NLnet.

Promotiecommissie:
prof.dr. F.M.T. Brazier (promotor)
dr. B.J. Overeinder (copromotor)
prof.dr.ir. H.E. Bal
dr. F.P.M. Dignum
prof.dr. C.M. Jonker
prof.dr. M. Luck
dr. J. Onvlee
dr. O.F. Rana

Cover design by Ana Pascha
Printed by PrintPartners Ipskamp, Enschede, The Netherlands

VRIJE UNIVERSITEIT

Agent-Based Mediated Service Negotiation

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. L.M. Bouter,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der Exacte Wetenschappen
op maandag 21 mei 2007 om 15.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

David Gillis Age Mobach

geboren te Amsterdam

promotor: prof.dr. F.M.T. Brazier
copromotor: dr. B.J. Overeinder

Contents

Preface

This thesis is one of a range of theses currently being written within the relatively new Intelligent Interactive Distributed Systems group at the Vrije Universiteit Amsterdam. As one of the first PhD students to enter the group in 2001, I learned many valuable lessons over the years witnessing the development of the group and its research, and learned even more from the people in it and their attitudes towards research. In this respect I am grateful to Frances, Benno and Niek, who proved to be experts in guidance, advice, support and inspiration throughout the years. I also thank my fellow PhD students Sander, Elth, David, Guido, and Reza, whose company I enjoyed very much. In addition I thank my other colleagues Michel, Renier, Martijn, Etienne, Olivier, and all others for making my stay at the Vrije Universiteit interesting and fun. Last but not least, I thank my friends, family and especially Marjolein, for building and maintaining the foundations that made it possible for me to spend so much time and energy on my research and this thesis.

Chapter 1

Introduction

1.1 Overall Research Context

The agent paradigm allows for the specification of computer systems as collections of autonomous and distributed components. Jennings et al. [50] specify a number of key characteristics of multi-agent systems:

- individual agents have specialized, limited problem solving capabilities and viewpoints;
- decentralized data;
- lack of global control;
- asynchronous computation.

An advantage of these characteristics is that within a multi-agent system, individual agents can be modeled relatively straightforward, as their tasks and interactions with the world are limited. However, due to the asynchronous nature of multi-agent systems, and the lack of global control, communication and coordination mechanisms are very important elements of agent-based systems. These provide the necessary interaction structures between individual agents, enabling exchange of information, and coordination of complex tasks involving many agents. Although not mentioned in the above list of characteristics, *mobility* is considered an important characteristic in *mobile agent systems*, an important subclass of multi-agent systems. Allowing agents to migrate has a number of advantages: Agents can move (closer) to machines where required data is stored, avoiding network latency and bandwidth limitations; Increased data protection as agents can access data locally. Lange and Oshima describe several other advantages in [59]. If designed and implemented correctly, agent-based systems can offer robust and efficient solutions in complex information system environments.

In recent years, applications of agent-based systems in real-world environments have emerged, particularly in areas where distributed planning/scheduling, or the management of distributed interactions, are important issues:

Magenta's Ocean i-Scheduler[1] uses agents in real-time planning of cargo assignment to vessels in a fleet. WhiteStein's *Living Systems Adaptive Transportation Networks*[2] (LS/ATN) uses agents to optimize the logistics of large-scale transport companies, taking into account the many constraints on their vehicle fleet, cargo, and drivers. *Acklin BV* developed an international vehicle insurance claims processing system (KIR) to connect companies from Belgium, the Netherlands and Germany, with business rules and logic encoded into discrete agents representing the data sources of the different companies involved.

Eurobios has provided an agent-based modelling solution [30] for *SCA Packaging* in order to explore different strategies for reducing stock levels without compromising delivery times, as well as evaluating consequences of changes in the customer base. The agent-based simulation developed by Eurobios allowed the company to reduce warehouse levels by over 35% while maintaining delivery commitments.

Working with US-based software developer Infotility, *CSIRO* has completed the first release of the *GridAgents* software framework[3], and is building a demonstration system at the CSIRO Energy Centre in Newcastle, Australia, putting a gas micro-turbine, photovoltaic arrays, and a wind generator under agent control, along with two cool rooms and a zone of a building climate system. This will form a mini-grid, coordinating supply and demand and reacting intelligently to electricity market or retail contract price signals.

HealthAgents[4] plans to create a multi-agent distributed decision support system, to help in the early diagnosis and prognosis of brain tumors. For more information on these applications, and other examples of commercial agent-based applications, see [12, 62].

Research in the area of agent-based systems can be classified as follows: Firstly, multi-agent system *(formal) design frameworks* are being developed. These frameworks are very diverse, ranging from modeling languages aimed at software engineering aspects of designing multi-agent systems, such as AUML [70], to organizational modeling frameworks such as GAIA [98] and DESIRE [19], that are aimed at modeling and formalizing knowledge and interactions within multi-agent systems.

Secondly, *agent programming languages* are being developed, such as 3APL [46], and AgentSpeak [79]. These languages offer constructs taken from imperative and/or logic programming languages, extended with agent-based programming concepts, such as cognitive agent models (e.g. BDI-based models) and agent interaction protocols.

Thirdly, *multi-agent system platforms* are being developed, providing basic infrastructures for agent-based applications. Well-known examples of such systems are JADE [13], Ajanta [51], Cougaar [45], Grasshopper [11], and Aglets [60]. These architectures provide agent run-time environments and libraries implementing basic agent models, communication and coordination protocols, and (in some cases) support for agent mobility. In addition to these 'general-purpose' agent platforms, there are platforms specifically developed for certain application domains, such as virtual markets (e.g. GEMS [7], MAGNET [27]), and simulations (e.g. Soar [99], Swarm [67]).

[1] http://www.magenta-technology.com/products/ischedulers/oceanfleet
[2] http://www.whitestein.com/pages/solutions/logistics/ls_atn.html
[3] http://www.ict.csiro.au/page.php?did=225
[4] http://www.healthagents.net

It is mainly the third category in which developments have lead to the adoption of agent-based systems in real-world application domains, ranging from telecommunication networks [76, 17] to electronic commerce [24, 57], and from transportation planning and stock management [31, 36] to distributed energy management [48, 94]. Although systems in this category generally are not based on the advanced specification and formalization features of the first two categories, they do provide the necessary structures for developing and deploying agent-based applications in real-world environments.

Agent platforms are considered a cornerstone technology for future generation service-oriented computing infrastructures [62]. These infrastructures require intelligent and distributed management solutions, capable of dealing with distributed, dynamic and heterogeneous environments securely. Current multi-agent platforms do not provide the required management features to operate effectively in these environments.

Management Challenges

A number of the challenges that next-generation agent platforms face can be characterized as *management challenges*. Due to the scale and diversity of the applications these platforms must support, traditional management approaches fall short. The following management challenges can be distinguished:

1. Managing Agents

Next-generation agent platforms need to support a very diverse agent population, ranging from small, simple agents (e.g. agent-based simulations) to large, complex agents (e.g. personal assistant agents, (mobile) Internet search agents). Furthermore, when dealing with mobile agent systems, migration support is needed to enable agents to move from one location to another, to access remote resources. To manage a heterogeneous agent population, a common *life cycle model* needs to be implemented in agent platforms, allowing platforms to perform basic management operations (e.g. creation, suspension, migration, and removal of agents).

2. Managing Multi-Agent Communication and Coordination

Different types of multi-agent applications have different requirements concerning communication and coordination. These requirements range from basic support for exchanging simple messages, to support for semantically rich agent communication languages and interaction protocols. With respect to communication, agent platforms need to provide robust and scalable mechanisms, capable of supporting these different communication models. With respect to coordination, next-generation agent platforms need to provide implementations of well-known coordination models such as auctions, markets, and brokering. For more detail on coordination models, see Omicini et al. [72] and Deugo et al. [33].

3. Managing Agent Resource Access

Agents interact not only with other agents. They also use *resources* offered by and through the agent-platform, by which they are supported, to perform their tasks. These resources include low-level resources, such as computational resources, as well as higher-level services, such as access to functionality offered by the agent system (directory services, application specific services), or by external parties (e.g. web services).

In next generation agent platforms, large numbers of agents perform their tasks concurrently, and compete with each other for access to limited resources. At the same time, owners of these resources need to regulate access, for reasons such as security, accounting, and performance. An important (and complicating) factor is that agents and resources are distributed across multiple (administrative) domains, each with different management policies concerning agents and resources. Next generation agent platforms need to offer agents coordinated and regulated service access across multiple distributed domains.

Of these three management challenges, the first two receive the most attention in current agent platform research and development: Most current platforms define an agent life cycle model, and offer implementations of various communication and coordination protocols. However, these models are often proprietary, and may not be suitable for use in application domains other than for which they were designed.

Standardization efforts in these areas are very important, as these provide the management structures and approaches that will be implemented in the upcoming generation of agent platforms. The FIPA[5] specification incorporates models for managing agents and agent applications, and includes (among others) an agent life cycle model, and specifications for various interaction and communication models, such as Contract-Net and brokering protocols. Also, agent communication standards such as message structures and message exchange protocols are specified. FIPA has recently been accepted as a new standards committee by the IEEE[6].

Less attention has been given to the third management challenge however. Traditionally, allocation of resources within multi-agent systems is based on centralized approaches such as combinatorial auctions, which do not incorporate the distributed and decentralized nature of agents, resources, and the administrative domains in which they are situated. More elaborate negotiation infrastructures are necessary to:

- Allow individual agents and resource providers to engage in localized interactions for negotiating resource access conditions,
- and which allows the incorporation of resource access/management policies specified by both individual resource providers, as well as those spanning administrative domains which the resources providers are part of.

In a recent survey published by the *AgentLink Technical Forum Group on Multiagent Resource Allocation* (TFG-MARA) [25], an overview is given of the approaches currently being considered promising in the area of multi-agent resource allocation. In the survey,

[5] Foundation for Intelligent Physical Agents, http://www.fipa.org

[6] Institute of Electrical and Electronics Engineers, http://www.computer.org

a distinction is made between traditional centralized allocation procedures, and decentralized approaches. The two main decentralized approaches described in the survey are auctions and negotiation.

This thesis presents a negotiation framework, based on an interaction model for negotiation of service agreements between agents and the platform by which they wish to be supported. The agreements specify guarantees and obligations concerning the use of services, and are used to manage service access. Negotiation is not performed between the involved parties directly, but is performed through third party intermediaries, called *mediators*. The negotiation framework is based in part on the WS-Agreement specification [9]. The specification is extended to allow for mediated negotiation between consumers and service providers within agent-based environments, and explicit acceptance of agreements in the final stage of the negotiation process.

The framework provides for a *policy based approach* to resource access management: The framework is modeled to allow the implementation of service access policies into the negotiation model, at both individual service and group levels. Policies are considered to be domain-specific, and are defined and specified outside the negotiation framework. These policies can be instantiated in the negotiation framework as negotiation policies, using the extended WS-Agreement negotiation language. This approach allows for resource owners and domain administrators to incorporate the framework into existing policy specification infrastructures.

1.2 Research Focus

In this thesis, a negotiation framework is described and evaluated. Agents and providers of services (the term *services* is used instead of *resources*, as a more general term) interact with each other, with the goal of regulating access to these services.

A number of aspects of a distributed, agent-based environment in which the framework is to be applied have been identified in the discussion above:

- Large number of (mobile) agents and service providers;
- autonomy of agents and platform administrators;
- heterogeneity of agents and the platform support infrastructure (i.e. hosts, operating systems);
- dynamics of both agents and platform infrastructure.

Based on these aspects, a number of requirements can be formulated for the negotiation framework, which must be met in order for the framework to cope with each of the environmental aspects, with the aim of providing an effective negotiation infrastructure:

1. **Consistent service acquisition model**: A negotiation framework should be part of basic functionality provided by an agent platform, offering a uniform method for establishing agreements throughout the distributed infrastructure, alongside other primary functionality such as agent-agent communication and agent migration. It

should be part of the basic *service acquisition* infrastructure, offering a consistent model for managing service access for all involved parties.

2. **Domain-independent and flexible**: A negotiation framework should be domain-independent and flexible, allowing many different agent applications and providers of services to use the framework in their respective domains of application.

3. **Straightforward to use**: A negotiation framework should be straightforward in its use, and should provide as much functionality as possible to help agents and service providers in establishing agreements. The parties should not be burdened by the framework, but gain from its use.

4. **Suitable for dynamic negotiation environments**: A negotiation framework should be able to cope with the dynamic nature of an agent-based environment. Agents and service providers may join and leave at any time. Moreover, individual agents and service providers may change their requirements regarding service access over time.

5. **Incorporate both local and domain-wide service access policies**: A negotiation framework should ensure that domain-specific policies regarding service access and use can be expressed in the framework, and are represented in the agreement establishment process: We distinguish between so-called *organizational domain-wide policies*, and *localized service provider policies*. Organizational policies specify policies which hold for all available services within a domain, and are used for defining general service access an usage conditions (e.g., fairness of service access distribution across the agent population, overall service access performance conditions, etc.). Local service provider policies are specified by individual service providers and concern individual services (e.g., service acces control in order to deny access to specific agents or to maintain service performance under heavy load).

   The above policies influence the use of services by agents, and should be taken into account during the process of establishing agreements.

Based on these requirements, a number of research questions are formulated. These indicate issues regarding the design of the framework for which solutions must be provided. Below, the four main research questions that are addressed in this thesis are presented.

Q1:*Can the negotiation framework support negotiation interactions over a wide range of service domains?*

Q2:*Can the negotiation framework provide a uniform negotiation architecture, suitable for application in these different service domains?*

Q3:*Can the negotiation framework incorporate both organizational and local service access policies in the negotiation process?*

Q4:*Can the negotiation framework accommodate for dynamic and heterogeneous agent and provider populations?*

The above research questions are used to guide the design of the negotiation framework, and the implementation of the framework in the AgentScape agent platform: The research presented within this thesis is also strongly related to the *AgentScape* project. AgentScape [96] is a distributed agent platform developed by the *IIDS group* at the *Vrije Universiteit Amsterdam*. The platform is designed to be scalable and secure, and offers support for heterogeneous and mobile agents. The work in this thesis contributes to the AgentScape platform by adding service negotiation functionality to the AgentScape implementation. In return, the AgentScape system provides a valuable testbed which can be used for the evaluation and testing of the work presented in this thesis. In the concluding chapter of this thesis, the four questions are revisited, and the solutions presented in the framework are discussed.

1.3 Main Contributions

The main contributions of this thesis are:

- A negotiation framework for service access has been designed and implemented which is based on the WS-Agreement specification: The basic protocol of this specification has been extended to allow for more negotiation expressiveness and flexibility. Policies can be defined which govern negotiation interactions at key points in the process.
- The negotiation framework has been deployed in two different application domains: Distributed agent middleware, and distributed energy management. Different negotiation policies have been implemented within the negotiation framework to achieve different negotiation interaction patterns required for each of the application domains. The negotiation framework has been implemented as part of the AgentScape agent middleware, and was used for a number of experiments and simulations, in the context of both the agent middleware and the distributed energy management application domains.
- In the AgentScape agent middleware domain, low-level resource allocation policies which assign computational resources to individual agents (e.g. round robin resource selection, random resource selection) have been implemented and evaluated. Agents within an AgentScape environment negotiate with multiple locations offering resources required by the agents, and choose (i.e. migrate to) the location that offers the best resource conditions. AgentScape administrators specify resource access policies, which are reflected in the negotiation policies that govern which resources (and which resource amounts) are available to agents for negotiation. Experiments have been performed to evaluate the performance of the basic negotiation infrastructure, and the resource allocation behavior of the framework.
- In the distributed energy management domain, negotiation policies have been implemented which enable competition between individual energy providers offering energy resources to clients, and to enable competition between groups of energy providers, represented by mediators in the negotiation process. Simulations have been performed to evaluate the resulting negotiation behavior.

- Two extensions of the negotiation framework have been described in the context of the distributed energy domain, each describing how the framework can be extended to accommodate additional negotiation models. The first extension describes the implementation of an auctioning model using the negotiation framework. The second extension describes the incorporation of decommitment penalties in the negotiation framework.

1.4 Thesis Structure

The structure of this thesis is as follows:

Chapter 2 provides a literature overview of the relevant research related to the design and implementation of the framework: *Multi-agent systems* and *negotiation theory* are discussed. In addition, the field of *service-oriented computing* is also discussed, as this field faces management challenges similar to the challenges described in this chapter. **Chapter 3** presents the negotiation framework. First, an overview of the negotiation interaction model is presented, followed by a description of the mechanisms provided by the framework. **Chapter 4** presents the first of two *use cases* that are used to evaluate the framework. In the chapter, the applicability of the framework in the domain of distributed energy management is examined. **Chapter 5** presents the second use case: An implementation and evaluation of the framework in the AgentScape distributed agent platform. **Chapter 6** concludes the thesis by summarizing the work, and revisiting the research questions posed in Chapter 1.

Chapter 2

Related Work

This chapter gives an overview of relevant literature for this thesis. First, as the framework presented in this thesis is centered around a negotiation model, an overview of relevant negotiation research is given. We focus on negotiation models in the context of agent-based systems, as our framework is agent-based. After this, the field of Service Oriented Computing is discussed. In this field of research, similar management challenges as presented in the previous chapter are distinguished. Negotiation models and agent-based approaches are considered an important part of solutions to these problems, and standards are being defined by the service oriented computing community to structure service negotiation processes, based on existing negotiation models. Finally, the relevance of the topics that have been discussed for the framework presented in this thesis is discussed.

2.1 A.I. and Negotiation

Negotiation is a process in which two or more interdependent participants, each with their own possibly incompatible goals, explore their options, bargain and strive for a mutually acceptable outcome. In its broadest form, negotiation can be viewed as communication [95]: exchanging proposals/preferences, asking questions, argumentation, can all be part of negotiation processes. In the field of *automated negotiation*, negotiation is commonly described in the context of software agents [49]. In this context, negotiation can be seen as a coordination model, enabling agents to interact with each other in a structured way. Coordination models, such as negotiation, enable agents to form virtual organizations to achieve common goals in shared environments. Dignum et al. [35] describe the role of coordination models in modeling virtual organizations. In the paper, coordination models are described as a common ground for incorporating concepts from organizational theory (e.g. roles, norms, goals) in the design of multi-agent systems.

2.1.1 Forms of Negotiation

In automated negotiation, two main forms of negotiation can be distinguished: *Auctions* and *bargaining* [88]. The main characteristics used to classify negotiation models are

the *number of participants* involved in the negotiation process and the *number of issues* that are subject of the negotiation process. With respect to the number of participants, negotiation models can be classified as *one-to-one*, *one-to-many*, and *many-to-many*. With respect to negotiation issues, negotiation models can be classified as *multiple-issue* or *single-issue*. See Bichler et al. [15] for more detail. In addition to *buyers* and *sellers*, negotiations may also include intermediaries: Trusted impartial parties that participate in the negotiation process, for example to provide fair negotiation solutions when the other parties cannot reach an agreement. The following sections will briefly discuss auctions, bargaining, and mediation.

Auctions

Participants in auctions make their preferences known to a centralized auctioneer via a one-way offering mechanism known as *bidding* (different bidding protocols exist, e.g. English, Dutch and Vickrey auctions [53]). The auctioneer determines the outcome of the negotiation process. Auctions are traditionally used in single-issue situations, where *price* is the issue to be negotiated. Multi-attribute auctions allow for simultaneous bidding on multiple attributes (e.g., volume and price), and are currently subject of research [58]. David et al. [32] describe a number of issues contributing to the complexity of multi-attribute auctions: determining what the winner criterion should include; how to formulate a bid considering the multiple attributes; what should the auctioneer reveal at the start of an auction. In this thesis, auctions are not further discussed, as the negotiation model presented in this thesis is based on mediation.

Bargaining

In bargaining, participants most often negotiate by exchanging proposals/preferences. Proposals are evaluated by participants using some form of *utility* determination function. Bargaining models consist of two main parts: Interaction protocols and strategies. Early models focus mainly on protocol aspects. An early and well-known bargaining protocol is the Contract Net protocol [87]. The basic Contract Net protocol distinguishes two roles for participants: *managers* and *contractors*. In this protocol, managers announce a task to contractors. Contractors respond by making an offer to a manager. The manager allocates the task to a contractor. Finally, the contractor confirms the allocation. Counter proposals are not supported in the original Contract Net protocol. However, the protocol has been extended in many ways, allowing for example negotiation over bundles of resources, as presented by Sandholm [83].

More recently however, negotiation models are being developed that incorporate both protocols and internal models for the negotiation participants: Faratin et al. [38] present a negotiation architecture for agents that is related to the Contract Net approach, but extends this model by allowing more elaborate offer/counter-offer interaction sequences between agents. Furthermore, the architecture explicitly defines two deliberative mechanisms for participants in the negotiation process: *Trade Offs*, and *Issue Set Manipulation*. The trade off mechanism allows agents to search for offers which are of equal value to earlier contracts, but which may be of greater value to the other party. The issue set manipulation

mechanism provides solutions for determining which issues to remove or add from the current issue set under negotiation.

Multi-issue Negotiation is an important subclass of negotiation. In these negotiations, participants negotiate for multiple items in a single negotiation sequence. It is possible for participants in a multi-attribute negotiation to reach a 'win-win' state in the negotiation process (often a Pareto optimization process) [77]. From a complexity point-of-view, multiple-issue negotiations are challenging [25]: Determining utility functions for evaluating combinations of items is difficult, as different combinations of items may be valued very differently, depending on the preferences of the individual participants. Also, the number of possible solutions when dealing with combinations of items is much larger than in the case of single-issue negotiation.

In addition to negotiation protocols, negotiation *strategies* are also an important aspect of modeling negotiation. Two main approaches to negotiation strategies can be distinguished: *Game theoretical* approaches originating from the field of economics, currently embraced by the A.I. community, and approaches based on technologies developed in the field of A.I.. In the first category, negotiation mechanisms are developed and analyzed with the goal of reaching optimal negotiation outcomes for participants [82]. A distinction is made between *cooperative* and *non-cooperative* negotiation situations. Furthermore, *incomplete* and *complete* information situations are distinguished. Game theory approaches are criticized for often depending on assumptions which are difficult to maintain in practical situations (e.g. full rationality). Also, in the case of multiple-issue non-cooperative negotiation, the negotiation problem from a game theory point-of-view is computationally intractable: The search space in which to find an optimal strategy from the available strategies grows exponentially.

Approaches from the field of Artificial Intelligence (genetic algorithms, evolutionary search techniques) are used to efficiently find negotiation solutions in cases where reaching optimal solutions may be difficult or impossible. Chevaleyre et al. [25] and Gerding et al. [44] provide comprehensive literature overviews of negotiation theory in the field of multi-attribute negotiation.

Mediation

In addition to direct negotiation between participants, negotiation can also take place through intermediaries. These intermediaries can play one of three roles: that of Matchmaker, Broker, or Mediator [97]. A matchmaker's only task is to find matching negotiation participants based on preferences and offerings of these participants. After a match has been made the parties are introduced to each other, and the matchmaking process is finished. A broker is more involved in the negotiation process: Brokers not only match negotiation participants, they also intermediate all interaction between the parties. Mediators are similar to brokers, but have the additional ability to negotiate on behalf of participants, and provide a static contact point for a dynamic group of negotiation participants.

In mediated negotiation, participants negotiate with each other through a mediator. A mediator is able to direct the negotiation process by analyzing the preferences of the participants, and using this information to obtain a negotiation solution that satisfies both

parties. However, tradeoffs with respect to one-to-one bargaining models are a more complex interaction model, and potential trust issues, as participants need to communicate their preferences (partially) to a third party. In situations where negotiation participants cannot/do not directly know each other, or are not able to interact with each other for other reasons, a mediator-based model may be the only alternative.

2.1.2 MAS and Negotiation

Multi-agent systems are a natural approach to modeling and implementing negotiation systems, as they provide a structured environment for instantiating negotiation participants as multiple independent autonomous agents, and offer communication primitives for communicating negotiation information. In some cases multi-agent systems explicitly provide negotiation support by offering negotiation-based interaction mechanisms. The FIPA standard specifications provide several interaction protocols, such as the 'FIPA Contract Net Interaction Protocol'. JADE [13] (as a FIPA compliant framework) provides implementations of these protocols to agent developers for implementation of auctions and bargaining between agents. MAGNET (Multi AGent NEgotiation Testbed) [27], designed for analysis of market-based systems, also provides support for negotiation interaction protocols. Hindriks et al. [46] describe an extension of the 3APL agent language for the specification of a multi-stage negotiation protocol. Predefined negotiation protocols such as Contract Net, and auction protocols are provided in the ZEUS *agent component library* [69]. Although the approaches mentioned above provide similar negotiation mechanisms based on well-known negotiation protocols, the interaction mechanisms used to implement the protocols are specific to each system. Negotiation interactions with other agents running on other platforms are thus difficult to realize. FIPA provides standardized negotiation protocols, which could contribute to improving interoperability between platforms. These protocols however do not include mediator-based approaches.

2.1.3 Contract Negotiation

Negotiation in agent environments is often concerned with the allocation of tasks or resources between agents. In traditional negotiation models, negotiation issues are represented using simple data structures. Cardoso and Oliveira [22] state that generally, formal contracts are preferred when establishing relationships between unknown parties. Furthermore, they state that contracts are an alternative sanctioning mechanism, when non-legal sanctions, such as reputation mechanisms are insufficient to constrain opportunism. In *contract negotiation*, negotiation is used to establish a contract between negotiation participants. After a contract has been established, the terms agreed upon in the contracts can be monitored/implemented. Research in this area focuses mainly on formal contract representations [81]. Tan and Thoen [34] present a contract language (LCR) based on a branching-time logic. In the paper, contracts are used to specify commitments made by organizations or individuals with respect to interactions within an environment, allowing for the evaluation and verification of agent interaction. Dignum et al. [90] present a contract representation language (DocLog) for electronic contracting systems in the context of Internet trading. These contracts distinguish three layers, each describing the content

of the contract in a different way: The Data Layer to allow easy processing of contracts by Enterprise Resource Planning (ERP) systems; the Natural Language Layer for presenting contracts to humans and other text-related issues; the Semi-Formal Layer for allowing electronic contracting systems to process and reason about the contract. In addition to contract languages, alternative negotiation interaction protocols are also studied: Mathieu et al. [65] present a model in which persistent contracts are exchanged between participants, instead of individual messages.

2.2 Service Oriented Computing

Distributed computing aims to combine computational resources distributed across many computer systems into virtual organizations, and to provide users with tools for deploying applications across available resources. Grid systems aim to provide distributed computing environments in a transparent and uniform way, allowing straightforward deployment of applications across heterogeneous resources [42]. This section shortly describes Grid systems and discusses recent developments in the newly emerging field of Service Oriented Computing.

2.2.1 Grid Architectures

Although models and architectures differ, Grid architectures typically consist of components providing:

- Resource models: Data models for describing resources that are available to applications.
- Resource naming schemes: Global namespaces for resources, according to some organizational structure (e.g., hierarchical, flat).
- Resource discovery protocols: Mechanisms for applications to discover available and suitable resources.
- Monitoring and control of resources: Mechanisms to regulate resource access, for e.g. management and accounting purposes.
- Scheduling of tasks/jobs on the available resources: Elements of the architecture responsible for assigning resources to applications.
- Identity management mechanisms: Authentication and authorization mechanisms, to guarantee security and integrity of the resources.

Of these components, the scheduling component is the most central part, as it is responsible for performing the main task, consisting of allocating computational jobs to available computational resources. Many different approaches to scheduling can be distinguished based on various technologies such as economic/market models, machine learning, and predictive heuristic models. Krauter et al. [56] describe a number of taxonomies

to differentiate architectural models of Grid resource management systems (RMS), analyzing and comparing a number of well-known Grid architectures with respect to this taxonomy. Typically, in these systems, the Grid resource management system takes on a matchmaking or brokering role to allocate user tasks/jobs on/across available resources. One of the best known early Grid systems is Condor [78]. In Condor, entities which require or provide resources publish their resource requirements/offerings in so-called *ClassAds* (analogous to newspaper *classified advertisements*). A matchmaker service matches classads, and notifies the involved entities when a match occurs. Condor supports decentralized ownership of resources. Scheduling (i.e. matchmaking) in Condor however remains centralized. Another well-known Grid system is Nimrod/G [21]. Nimrod/G has been developed as an infrastructure to deploy parameter study applications across distributed computing environments, and is designed to use other Grid RMS systems such as Condor and Legion. Nimrod/G uses an economy based approach: Both deadline (soft-realtime) and budget (computational economy) constraints in scheduling are supported. Although Grid systems are successfully applied in many application areas, the goal of providing ubiquitous computing analogous to the power grid has not yet been attained. Recent developments in the field of Grid computing are aimed at further virtualization of the underlying Grid infrastructures, and standardization of the interfaces offered to Grid application developers. Examples of such systems are the Grid Application Toolkit (GAT) [8], the Simple API for Grid Applications (SAGA), [1] and the Open Grid Service Architecture (OGSA) [40].

2.2.2 Service Oriented Architectures

As Web Service architectures emerge, Grid architectures have adopted standards and practices developed in the web service community: So-called *Service Oriented Architectures* (SOAs) [43]. SOAs are architectures composed of loosely coupled *services*, interacting through standardized interfaces, modeled independently from underlying platforms. The *Open Grid Service Architecture* (OGSA) currently developed by the Open Grid Form (OGF) [40], is an attempt to provide standards for SOAs in Grid environments. OGSA defines a number of core web services providing basic Grid environment functionality (directory services, resource management services, scheduling and reservation services, etc.). The aim of OGSA is to provide a service-based environment, using web service interaction standards, in which additional Grid services can easily be deployed and accessed. Another example of a service-based architecture is the *Service Bus* described in [61]: A basic middleware layer (the *bus*) as a virtualization layer between users and services. Users request services based on high level criteria (WSDL, semantic descriptions, etc.), after which the service bus determines the best way to accommodate these requests. The bus hides implementation details of specific services, and allows for transparent composition (orchestration) of services.

An important concept within SOAs is the *Service Level Agreement* (SLA). SLAs are widely used in the information system domain to express Quality of Service (QoS) guarantees and obligations with respect to delivered products and services. Traditionally, these documents are written in natural language. In SOAs, SLAs are created between users and

[1] http://wiki.cct.lsu.edu/saga/

providers of web services, and describe the conditions under which a service is accessed by a user (job priorities, required computational resources, etc.). Established SLAs are often used by resource management services as a basis for making management decisions to ensure proper provisioning of agreed upon resources to users. In these service oriented environments, automated SLA management is desirable, as human intervention should be kept to a minimum. An example of a framework for automated SLA management is the Web Service Level Agreement (WSLA) [52] framework. This framework presents a language for specifying SLAs, and an architecture consisting of SLA monitoring services.

Negotiation in Service Oriented Environments

In the AgentLink Roadmap [62], negotiation research is mentioned as one of the main research challenges in the coming years in the development of service oriented architectures. With respect to SOAs and negotiation, the Roadmap states:

> ...bargaining over multiple parameters or attributes to establish SLAs between service providers and service consumers will be key in future service-oriented computing environments.

The Open Grid Forum's current aim is to standardize infrastructures for automated negotiation and creation of SLAs. The WS-Agreement specification [9] offers a basic contracting protocol and language, which can be used to establish agreements between service providers and consumers, based on predefined contracting templates. Although the specification provides a straightforward approach to establishing agreements, it offers a one-way interaction protocol, and does not allow for more elaborate negotiation interactions. Several attempts are being undertaken to extend the contracting protocol. Paurobally et al. [75] integrate Speech Acts into the model, allowing for a more expressive negotiation language, enabling more elaborate negotiations (e.g. Contract Net protocol). Aiello et al. [5] add *re*-negotiation to the protocol. The Open Grid Forum itself is working on an additional specification, WS-Negotiation. This additional layer on top of WS-Agreements will offer concepts for the specification of negotiation protocols based on the WS-Agreement primitives. For more information and a more elaborate discussion of the aforementioned WSA specification and extensions, see Chapter 3.

2.2.3 Agents and Grid Resource Management

The AgentLink Roadmap indicates that as the complexity of service-oriented environments increases, software agents can play an important role in these environments, both as a design paradigm, and as an enabling technology, for example to implement automated SLA negotiation infrastructures.

The use of the agent paradigm in Grid systems is not a new development. An early example of a Grid system in which the agent concept is used is Netsolve [23]. In a Netsolve system, agents are responsible for maintaining resource information, and act as brokers between users and computational resources. Recent developments in this area include more elaborate agent-based approaches, such as *CONOISE-G* [68], an agent-based architecture intended to implement virtual organizational formation and operation in a service

based environment. Another well-known system is Planetlab [26], in which agents track resources, and interact with resource brokers by advertising resources and issuing tickets with which resources can be claimed. Foster et al. [41] discuss the convergence of Grid and agent research communities, and highlight a number of important areas of research, in which both communities have an overlapping interest:

- Autonomy of system components: Both communities require models for creating systems consisting of independent components, capable of performing their tasks in a heterogeneous and dynamic environment.

- Negotiation and contracting: The use of negotiation to establish service contracts is considered important in both communities. Agent researchers have already provided significant contributions in this area, as described earlier in this chapter.

- Management of virtual organizations: Using organizational models to structure systems is also considered important in both communities. Current grid systems allow for describing and grouping services, but require higher-level semantic discovery and brokering functionality. Work performed in the agent community can be incorporated to provide this functionality.

- Authentication, trust, and policy management: Due to the dynamic and open nature of systems created in both communities, models for authentication of dynamically created entities are important, as well as models for establishing trust relationships between entities, and for policy specification and enforcement within these environments.

2.2.4 Multi-Agent System Resource Management

As (mobile-)agent-based systems are moving out of academia and into real-world applications, performance requirements are becoming a more pronounced concern. Multi-agent systems are being deployed on existing networking and computing infrastructures: Mechanisms are required to allow monitoring and control of agent resource consumption. Work in this area ranges from low-level monitoring and enforcement mechanisms that are used internally for management purposes, to infrastructures that allow agents to interact with management systems about resource requirements and availability.

Binder et al. [16] describe a resource aware version of the Java based J-SEAL2 mobile agent kernel, which is aimed at preventing denial-of-service attacks by agents through regulation of agent resource usage. The infrastructure allows for the regulation of agent (relative) CPU consumption, active memory, and active and total number of threads. Sharing of allocated resources by agents is also supported and is also subject to regulation.

In the NOMADS [89] multi-agent platform, fine-grained resource control of disk and network access (rate and quantity) is achieved using mechanisms provided by a Java-compatible Virtual Machine (Aroma).

Ajanta [51] is a mobile agent platform in which resource access is governed by policies that can be defined and implemented by resource owners. Resource access is provided to agents through *proxies*, enabling the embedding of metering and accounting mechanisms.

2.3 Conclusions

In this chapter, an overview of different agent-based negotiation approaches is given. Although implementations of these models are available, these are mostly application specific. Although FIPA is offering basic negotiation protocols as part of its standardization effort, these protocols are oriented towards coordination of tasks, and are not intended as a basic interaction mechanism as part of the basic platform infrastructure. For agent-based negotiation models such as those discussed in the beginning of this chapter to be successfully implemented in large-scale and heterogeneous environments, a common and standardized framework needs to be created which is able to support these models. Such a framework can provide a common base for all users and providers of negotiation systems built op top of this framework, and should consist of the components as indicated earlier in Section 2.1.1: A negotiation language, negotiation protocol, and generic internal agent models for the negotiation participants, including support for the specification and implementation of domain-specific negotiation strategies. Furthermore, we argue that such a generic framework should be based on a *mediated* negotiation model, as this provides the framework with a number of useful properties that are of importance in large-scale and heterogenous domains: Mediators function as relatively static contact points for negotiation participants in the dynamic environment (participants may not be able to known each other directly due to the size of the environment/population); mediators can acts as trusted parties when negotiating with unknown parties; mediators can translate between different parties. Finally, to allow for the expression of complex negotiation issues in a generic manner, and as formal agreement structures are preferred in situations with unknown negotiation participants, a well-defined contract-like information structure should be provided, in which domain-specific negotiation issues can be integrated with minimal effort.

The overview of the field of Service Oriented Computing given in this chapter indicates that interaction models for establishing agreements have been and are an important issue in this field: Grid systems introduced matchmaking services for job/task scheduling, which have evolved to more elaborate brokering services allowing high-level deployment of applications on heterogeneous distributed infrastructures. More recently, as the field moves in the direction of higher-level general Service Oriented Architectures, frameworks are needed that allow automated negotiation of service level agreements between service providers and service consumers, to reduce the time consuming intervention of humans in this process. The initiatives that are currently being researched (e.g., WS-Agreements) are promising, but are only able to model very basic interaction models.

2.3.1 Framework Positioning

The negotiation framework presented in this thesis originates from the need of both fields for the specification and implementation of service negotiation, in a generic and standardized way.

For the field of agent-based negotiation, the framework offers the possibility of implementing the various negotiation models that have been, and are still being researched, using a standardized basic interaction model and language. The framework can then be

offered to agent application developers as part of a larger agent-based middleware system (such as the AgentScape middleware described in Chapter 5), offering a basic support infrastructure for specifying, implementing, and using negotiation models.

For the field of service oriented computing, the framework offers an extension of the specifications and frameworks currently being developed. The extensions allow for more realistic negotiation models to be expressed, which can in turn provide more solid foundations for automated service level agreement negotiation systems. Domain-specific negotiation policies can be specified which tailor the negotiation framework to the specific negotiation requirements in different domains.

To summarize, the framework can be viewed as an attempt to bridge the gap between service oriented computing and agent-based computing, using negotiation as a common denominator between the two fields. Negotiation is a key component of future generation distributed computing architectures, and agent-based architectures play an important role in future service oriented environments, to provide autonomous and distributed negotiation infrastructures. The negotiation framework described in this thesis aims provide an agent-based mediation approach for service negotiation in distributed environments.

Chapter 3

The Negotiation Framework

3.1 Introduction

This chapter presents a negotiation framework within which consumers negotiate with providers for service access through mediation. Mediators present consumers with aggregations of services, available within the *virtual organization* it represents. In the framework, providers are assumed to be aggregated in virtual organizations: Providers can form dynamic groups, based on criteria which may or may not relate to the services they offer. The creation and management of these virtual organizations is performed outside the negotiation framework. Mediators are dynamic: At any point in time can a mediator be assigned to represent a virtual organization in a negotiation process. This mediator is then responsible for handling negotiations between service providers within the virtual organization, and any service consumers. During negotiations, mediators enforce organization-wide service policies. Negotiation is one-to-many: Consumers can enter into negotiations with multiple mediators. Mediators negotiate with multiple service providers simultaneously. The environment is dynamic: Consumers can interact with multiple mediators concurrently, and services and service providers can join and leave virtual organizations at any time. Interactions between parties are based on the WS-Agreement [9] negotiation protocol and language. Extensions to the protocol are made to accommodate for mediated negotiation. Section 3.2 identifies the roles of the participants and introduces the negotiation model. Section 3.3 describes the negotiation language used in the framework. Section 3.4 defines the negotiation protocol. Section 3.5 provides an overview of the task models of each of the modeled agents. Finally, Section 3.6 presents a summary and discussion of the framework.

3.2 The Negotiation Framework

This section presents an overview of the negotiation framework. Requirements for the negotiation framework have been defined in Section 1.2. Based on these requirements, framework design decisions are made. These decisions are described below:

- Standardized negotiation language and protocol:
  The protocol and language are based on an emerging agreement negotiation standard (WS-Agreements). This standard provides a domain-independent agreement negotiation language (*requirement* 2), and a straightforward interaction protocol to establish agreements (*requirement* 1).

- Single cycle negotiation:
  To prevent lengthy negotiations, negotiation interactions consist of a single request-offer sequence. Advertisements are used to limit the negotiation space at the start of each negotiation. This relatively simple negotiation scheme ensures that participants do not have to manage complex negotiations consisting of multiple negotiation cycles involving revision of requests or offers (*requirement* 3).

- Mediated negotiation:
  A mediated negotiation model is chosen as a basis for the framework. The use of mediators hides the dynamics of the participant population in the negotiation process (*requirement* 4). Also, mediators can provide the means for implementing domain-wide service access policies during negotiations (*requirement* 5).

Policy Based Negotiation

The framework is designed to be policy based: The behavior of the negotiation framework is governed by negotiation policies that are specified for each negotiation domain in which the framework is implemented. These negotiation policies are based upon external, domain-specific policies: We assume that each domain has its own policies regarding access to, and use of available services. These are specified outside the negotiation framework, and may be part of some more general policy infrastructure. To achieve the goals as specified by these external policies, they are translated into negotiation policies which govern the behavior of the components of the negotiation framework. Figure 3.1 shows the relation between the different policy levels. In the components of our framework, several types of negotiation policies are distinguished. These are introduced and discussed in more detail throughout this chapter.

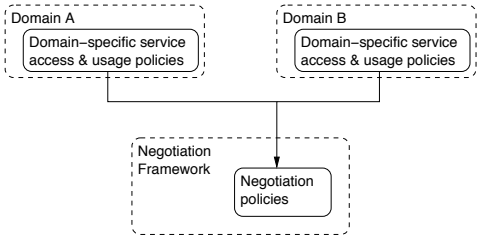

Figure 3.1: Negotiation policies.

Allowing for the incorporation of different negotiation policies in the negotiation framework means that the implementations of the agent roles defined by the framework can be different in different domains. For example, in one domain, a mediator agent can be configured to act with the interests of the consumer in mind, while in another domain a mediator may act in favor of providers.

3.2.1 Participants

The framework distinguishes consumer agents, mediator agents and service provider agents:

Consumer Agent

Each consumer agent represents one consumer in a negotiation process. Consumer agents initiate a negotiation process, aiming to acquire an offer which they can accept: An offer that fulfills their requirements.

Service Provider Agent

A service provider agent represents the provider/owner of one or more services in negotiations with a mediator agent. Access and usage policies are defined, implemented, and enforced by the provider/owner of each service individually. Each service provider agent is a member of exactly one virtual organization of service provider agents.

Mediator Agent

A mediator agent represents a virtual organization of service provider agents in negotiations with consumer agents. Each virtual organization has its own negotiation policies spanning the service provider agents involved. A mediator agent is responsible for the implementation and enforcement of these organization-wide negotiation policies.

3.2.2 Negotiation Model Overview

Consumer Agents (CA) negotiate with mediator agents (MA). Mediator agents negotiate with service provider agents (SPA), on behalf of CAs. Figure 3.2 presents an overview of the model.

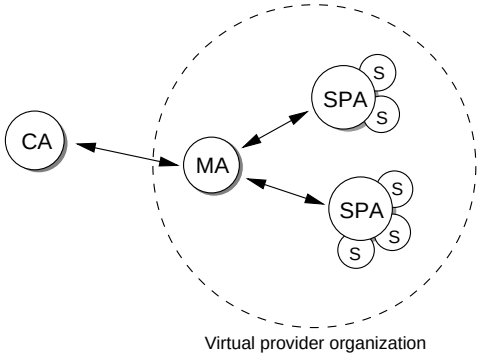

Figure 3.2: Negotiation model overview.

Our negotiation model is two-tiered: In the upper tier, consumer agents negotiate with mediator agents. Mediator agents hide provider-level details from consumer agents, offering a single point of contact. In the lower tier, mediator agents negotiate with service

provider agents. Mediator agents translate negotiation requests from consumer agents into requests for services for service provider agents.

Mediator agents combine information on the availability of services within a virtual organization at a specific point in time. This information is provided by service provider agents in the form of advertisements. Consumer agents bid for these services. Mediator agents determine an optimal allocation across the services offered by the individual service provider agents once a bid has been made, according to the policies of the virtual organization.

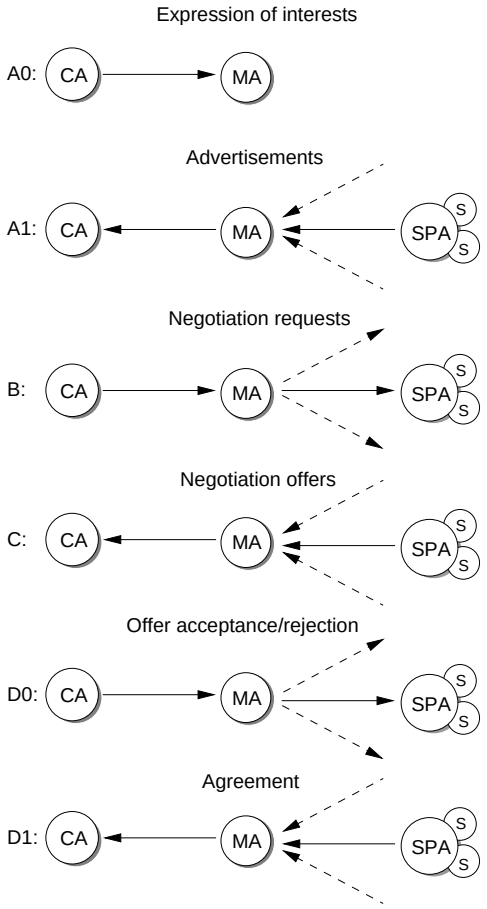

Figure 3.3: Negotiation interactions.

The negotiation interactions used in both tiers consist of simple request-response type negotiation interactions. Figure 3.3 gives an overview of the negotiation interactions in the negotiation process. The negotiation process consists of four main phases, extending the WS-Agreement protocol :

Advertisement phase (A):

In this phase, consumer agents request advertisements from mediator agents. Con-

sumer agents express the services in which they are interested in the request (A0). Mediator agents collect and combine individual service provider agent advertisements into combined advertisements. Any combined advertisements that contain services that the consumer agent is interested in, are returned to the consumer agent (A1). Note that the retrieval of service provider agent advertisements is a separate process, which can occur concurrently, independent of the consumer agent advertisement request process.

Request phase (B):
Consumer agents create negotiation requests based on advertisements. Mediator agents translate these requests into negotiation requests for individual service provider agents.

Offer phase (C):
In the offer phase, service provider agents return negotiation offers in response to the received negotiation requests. Mediator agents combine and select the best offer(s), and return this to the consumer agent.

Acceptance/rejection phase (D):
In the acceptance phase, the final agreement is established. Consumer agents accept (or reject) the received offer(s). Mediator agents translate this into acceptance/rejection of individual service provider agent offers (D0). Service provider agents return the established (partial) agreements to the mediator agent. The mediator agent combines the partial agreements into a single agreement, and returns this to the consumer agent (D1).

The format of the negotiation information exchanged is based on *agreement document standards* as described in the Web Service Agreement specification (WS-Agreement) [9]. WS-Agreement template documents are used to start a negotiation process and to structure negotiation sequences: During a negotiation sequence, parties exchange agreement documents, resulting in a final agreement document the parties have agreed upon.

Role of the Consumer Agent

Consumer agents perform multi-attribute negotiations with one or more mediator agents, the goal being to acquire the right to service access. Consumer agents are assumed to have already located the mediators with whom they wish to negotiate, through some external means (e.g. directory services). The negotiation protocol described in this chapter focuses on interactions with one mediator agent, but the framework allows for multiple negotiations concurrently. Agents may need to negotiate with different mediators in order to fulfill their full set of requirements. Consumer agents are assumed to be capable of translating their domain-specific requirements into negotiation requests for services using the negotiation language as specified in the framework. Similarly, agents are assumed to be capable of interpreting the negotiation offers they receive.

Role of the Mediator Agent

A mediator agent collects and combines service provider advertisements, and provides a single point of access for consumer agents to negotiate for multiple services provided by different service provider agents. In the request phase, a mediator agent forwards (parts of) a request to selected service provider agents. Domain-specific utility calculation mechanisms determine which providers can potentially provide the best offers, based on their advertisements and local policies. In the next phase, the offer phase, a mediator agent compares the offers these providers return, again using the aforementioned utility calculation mechanisms to determine which offer(s) should be forwarded to the consumer agents. The negotiation process ends with an agreement.

Role of the Service Provider Agent

Service provider agents provide a contact point for mediators to negotiate for access to services. Service provider agents specify advertisements that represent an up-to-date and accurate view on the current state of services and policies. All incoming negotiation requests are based on these advertisements. The content of the advertisements is based upon availability and expected utility of services, and local policies concerning service access. During negotiations, service provider agents create negotiation offers in response to requests from the mediator agent. The content of an offer is based on (i) the content of the request, (ii) the availability of the requested services, and (iii) local policies regarding service access. Service provider agents are responsible for ensuring that the services they have offered can be delivered if the negotiation succeeds (i.e. commitment to the offer made). Domain-specific *service reservation mechanisms* may be needed.

After an agreement has been negotiated, service provider agents are responsible for enforcement of the agreement.

3.3 The Negotiation Language

The negotiation interactions between the agents presented above are based on the Web Service Agreement (WS-Agreement) specification. Below a description of the WS-Agreement Specification is provided, followed by a description of the extensions made to the specification, and the use of the specification within our negotiation framework.

3.3.1 Web Service Agreement Specification

The WS-Agreement specification [9] describes a domain-independent negotiation-based approach for accessing web services. To obtain access to and use a web service, a negotiation cycle takes place between the two involved parties: A web service provider and a web service consumer.

Service providers specify the services they offer and the conditions under which the services may be used by consumers in a *template* document. Service consumers may request such a template document and use it to formulate an *agreement request*, specifying the desired services and service conditions in the request document. Service providers

interpret the request and respond with an *agreement* document if the requested services and service conditions can be met. The resulting agreement specifies the conditions under which the service can/may be used by a consumer.

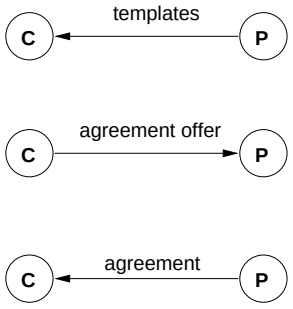

Figure 3.4: The WS-Agreement protocol.

The specification has been designed specifically for service oriented architectures, and defines a negotiation protocol and language. The interaction protocol is relatively straightforward (see Figure 3.4): First, the initiator of a negotiation sequence (C) requests available *templates* from the service provider (P). The initiator then selects a suitable template and uses this to create an *agreement offer*. This offer is sent to the service provider. If accepted, the provider creates an *agreement* based on the offer. The agreement is implemented, and returned to the initiator.

The specification defines negotiation document formats using XML-Schema: Two document types are distinguished: *Templates*, and *Agreements*. Central to these document types are negotiation *terms*. Service descriptions are explicitly not modeled, and can be added to the specification according to the requirements of the application domain. The following sections describe the document types in more detail.

Figure 3.5: Conceptual overview of an agreement.

Agreement Document

An agreement consists of two main sections: a *context*, and *terms* section. The context section contains agreement meta-information, such as identifiers for the initiator of an agreement, and the provider of an agreement, the name of the template on which the

agreement is based, references to other agreements (if present), and agreement *duration*. Relationships between agreements are not further defined in the WS-Agreement specification. Agreements are time-limited: The duration specified in the context section of an agreement indicates when the agreement expires. Figure 3.5 depicts the main fields in an agreement document.

The terms section contains the actual agreement content. Two types of terms are distinguished: *Service Description Terms* (SDTs), and *Guarantee Terms* (GTs). SDTs define the services involved. GTs refer to the described services, and define assurances to the consumer on service quality and/or resource availability offered by the service provider. Agreements can contain multiple SDTs, which can refer to multiple services. An example of an agreement document is shown in Example 3.1. In the example, an agreement established between provider P and consumer C for access to a video library service is shown. The agreement is based on template T, which was specified by P beforehand. The agreement is valid for one hour (duration = 3600s), and specifies a single *video_library* service. Also, the agreement specifies that the mediatype delivered by the library is MPEG-4. Finally, the agreement specifies a guarantee term, indicating that the minimum streaming bandwidth agreed upon by both parties is 250 Kb/s.

```
<Agreement>
 <Name>Agreement_Example</Name>
 <Context>
  <AgreementInitiator>Consumer C</AgreementInitiator>
  <AgreementProvider>Provider P</AgreementProvider>
  <Template>Template T</Template>
   <Duration>3600</Duration>
 </Context>
 <Terms>
  <All>
   <ServiceDescriptionTerm Name="sdt-1" serviceName="video_library">
    <mediatype>MPEG-4</mediatype>
   </ServiceDescriptionTerm>
   <GuaranteeTerm Name="gt-1">
    <ServiceScope>
     <ServiceName>video_library</ServiceName>
    </ServiceScope>
    <ServiceLevelObjective>
     <minimum_bandwidth>250</minimum_bandwidth>
    </ServiceLevelObjective>
   </GuaranteeTerm>
  </All>
 </Terms>
</Agreement>
```

Example 3.1: An agreement document.

Template Document

A template has a document structure similar to an agreement document, but with the addition of a *Creation Constraint* section, which is used to define initial negotiation constraints on service description terms described in the template (e.g. specifying a maximum or minimum value for a service request). Figure 3.6 depicts a template document.

The creation constraint section consists of *Items*. An Item is a restriction describing a field in the agreement (indicated using an XPath [14] expression in the *Location* tag), and

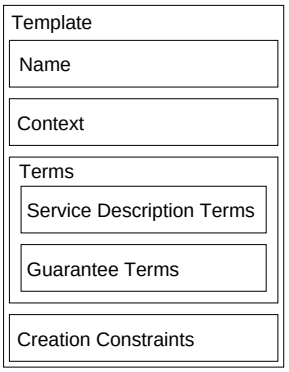

Figure 3.6: Conceptual overview of a template.

value restrictions for that field. Restriction models are not specified in the WS-Agreement specification, they are considered to be domain dependent. Creation constraints are only an initial indication of valid values for agreement requests: An agreement offer adhering to the constraints described in the template does not necessarily guarantee the acceptance of the agreement request. An example of an agreement template is shown in Example 3.2: The template with name T is specified by Provider P as indicated in the context section, and describes a single service *video_library*. The library delivers media of type MPEG-4. Furthermore, the template specifies a constraint on the value which may be requested for the *minimum_bandwidth* of the service, limiting this to a maximum value of 1500 Kb/s.

```
<Template>
 <Name>Template T</Name>
 <Context>
  <AgreementProvider>Provider P</AgreementProvider>
 </Context>
 <Terms>
  <ServiceDescriptionTerm name="sdt-1" serviceName="video_library">
   <mediatype>MPEG-4</mediatype>
  </ServiceDescriptionTerm>
  <GuaranteeTerm name="gt-1">
   <ServiceScope>
    <ServiceName>video_library</ServiceName>
   </ServiceScope>
   <ServiceLevelObjective>
    <minimum_bandwidth/>
   </ServiceLevelObjective>
  </GuaranteeTerm>
 </Terms>
 <CreationConstraints>
  <Item>
   <Location>
    //GuaranteeTerm[@name='gt-1']/ServiceLevelObjective/minimum_bandwidth
   </Location>
   <max_value>1500</max_value>
  </Item>
 </CreationConstraints>
</Template>
```

Example 3.2: A template document.

3.3.2 WS-Agreement Specification Usage

The negotiation language and protocol defined in the WS-Agreement specification are applied in our negotiation framework to provide the basis for the agent-based negotiation interactions. Both the protocol and language have been extended to allow for a more flexible negotiation framework. In this section, adaptations to the negotiation language are discussed. In Section 3.4, extensions to the negotiation protocol are discussed.

WS-Agreement Language Adaptations

In our framework, advertisements are specified using WS-Agreement *template* documents: Service providers use the service description terms and creation constraints to specify available services and negotiation restrictions.

Negotiation requests, negotiation offers, and final agreements are specified using WS-Agreement *agreement* documents: Service description terms are used to specify the services under negotiation. In our framework, services are *typed*: The *serviceName* attribute in the service description term indicates the type of service that is being described in the term. Instead of being specified in separate guarantee terms, service level guarantees are modeled as part of domain-specific service descriptions contained within the service description terms: In our framework, the distinction between what is to be part of a service description and what is to be part of a service level objective is not made at this (domain-independent) level in the negotiation language, but is instead considered part of domain-specific service descriptions. Example 3.3 depicts an agreement document as used in our framework.

```
<Agreement>
 <Name>Agreement_Example</Name>
 <Context>
  <AgreementInitiator>Consumer1</AgreementInitiator>
  <AgreementProvider>Provider1</AgreementProvider>
  <Template>Template1</Template>
   <Duration>600</Duration>
 </Context>
 <Terms>
  <All>
   <ServiceDescriptionTerm Name="..." serviceName="...">
    <!-- domain-specific service description,
    including domain-specific service level objectives -->
   </ServiceDescriptionTerm>
  </All>
 </Terms>
</Agreement>
```

Example 3.3: An agreement document as used in the negotiation framework.

Negotiation Restriction Model

A basic service access restriction model has been added to the WS-Agreement specification, allowing for the expression of basic constraints over services: *Maximum value* constraints, *minimum value* constraints, and *constraints enumerating allowed values*. Example 3.4 shows the syntax of the restriction model: Each restriction is defined using

a *Restriction* element. The *type* attribute is used to indicate the restriction type (*maxValue/minValue/enumeration*). Type specific elements are used to express the actual restrictions.

```
<CreationConstraints>
 <Item>
  <Location>//xpath-expression-to-service-description-term</Location>
  <Restriction type="maxValue">
   <maxValue>350</maxValue>
  </Restriction>
 </Item>
 <Item>
  <Location>//xpath-expression-to-service-description-term</Location>
  <Restriction type="minValue">
   <minValue>50</minValue>
  </Restriction>
 </Item>
 <Item>
  <Location>//xpath-expression-to-service-description-term</Location>
  <Restriction type="enumeration">
   <enum>AllowedValue1</enum>
   <enum>AllowedValue2</enum>
   ...
  </Restriction>
 </Item>
</CreationConstraints>
```

Example 3.4: The restriction model.

Term Composition

In our framework, term composition is an important aspect of the negotiation documents (both in advertisements and agreements): The mediator agent uses term compositors to manipulate documents (combining advertisements and requests/offers during negotiations, etc.). The WS-Agreement specification support term composition using term compositors taken from the WS-Policy specification [10]. WS-Policy defines three main composition elements: *All*, *ExactlyOne*, and *OneOrMore*, which are used as logical AND/XOR/OR operators. These operators are used to express policies concerning the combination of various services described in the service description terms.

The WS-Policy specification also specifies operations for *normalizing*, *merging*, and *intersecting* policy assertions. In our framework these operations are used to process service description terms in the negotiation documents. Normalization of assertions entails reducing the complexity of a nested set of assertions, to a *normal form* representation of the nested set. This form is subsequently used to create combinations and intersections of sets of service descriptions. The normal form of a document contains a single *ExactlyOne* element at the root of the specification. The alternatives within this element are encapsulated using *All* elements. Example 3.5 shows an example of a set of service description assertions in the normal form. In this example, the normal form expresses that either *serviceA*, or *serviceA* and *serviceB* together can be requested. The *merge* and *intersect* operations are discussed in more detail in the protocol description in Section 3.4. For more information on the processing models see the WS-Policy specification [10].

```
<ExactlyOne>
 <All>
  <ServiceDescriptionTerm serviceName="serviceA"/>
  <ServiceDescriptionTerm serviceName="serviceB"/>
 </All>
 <All>
  <ServiceDescriptionTerm serviceName="serviceA"/>
 </All>
</ExactlyOne>
```

Example 3.5: Normal form representation of service description terms.

Domain-Specific Service Descriptions

The WS-Agreement specification does not specify the actual content of the negotiation document, nor does our framework. This depends on the domain of application. The only requirement with respect to the negotiation issues in these domains is that they can be represented as *services* in the negotiation language, which is applicable to a broad range of domains. Examples of service descriptions in two different negotiation domains however are used in Chapters 4 and 5 to demonstrate the framework potential.

3.4 The Negotiation Protocol

In this section, the four negotiation phases of the protocol, introduced in Section 3.2.2, are described, from the perspective of each of the negotiation participants.

In our framework, the WS-Agreement protocol (see Section 3.3) is extended with a fourth negotiation phase. In its original form, the WS-Agreement protocol consists of three phases, in which the initiator of a negotiation implicitly accepts the returned agreement it receives from the other negotiation party. In our framework, initiators of negotiations need to be able to negotiate with multiple parties simultaneously. The WS-Agreement specification does not allow for this, as it assumes full commitment of the initiator of a negotiation: Agreement requests cannot be revoked in favor of better negotiation offers. To enable agreement initiators to conduct multiple negotiations for the same services with different service providers concurrently, an explicit acceptance/rejection phase is added to the protocol. This allows initiators to select between multiple offers, and select the best offer, instead of implicitly committing to multiple agreements, as in the original protocol. Figure 3.7 show the phases. Note the inclusion of the fourth negotiation phase.

3.4.1 Advertisement Phase

In the advertisement phase, consumer agents request the combined advertisements. Mediator agents combine advertisements of individual service provider agents.

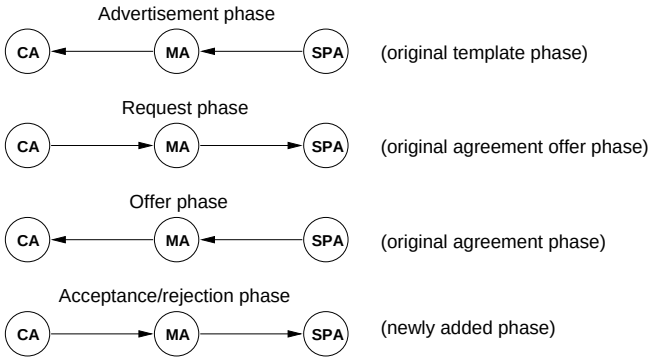

Figure 3.7: Overview of the negotiation phases in the protocol.

Advertisement phase: Consumer Agent

For a consumer agent the advertisement phase entails (i) formulating service interests, and (ii) communicating these interests to one or more mediator agents, which have been selected by the consumer agent. Consumer agents inform the mediator agents of the services in which they are interested, using domain-specific terms. For example, a consumer may indicate that it is interested in negotiation about access to two services: *diskSpace*, and *printingService*. The consumer agent indicates its interest in these services by specifying the service names in the interest list:

```
<Interests>
  <ServiceDescriptionTerm serviceName="diskSpace"/>
  <ServiceDescriptionTerm serviceName="printingService"/>
</Interests>
```

As a result, the consumer agent receives relevant advertisements describing the services and conditions available for negotiation from each of the mediators. A consumer agent can also communicate an empty list to a mediator agent, which results in the mediator agent returning all available advertisements. This phase could also include authentication: Authentication information can be passed from the consumer agent to the mediator agent along with the service interests described above. This aspect is subject of future research, and as such is not further explored in this thesis.

Advertisement Phase: Mediator Agent

In the advertisement phase, a mediator agent (i) retrieves and combines advertisement documents from service provider agents, and (ii) responds to requests for advertisements from consumer agents.

The mediator agent requests the available advertisements from the service provider agents within the virtual organization. These advertisements are then reduced to their normal forms, and the *merge* operation is applied to the documents to integrate the individual advertisements into combined advertisements.

When a request for advertisements is received, the service interests as specified by the consumer agents in this request are compared to the available services described in the

combined advertisements. Any matching advertisements (i.e., advertisements containing one or more services that match with the services indicated in the service interests) are returned to the consumer agent. If no direct matches are found, the mediator agent may attempt more elaborate (semantic) matching procedures, for example matching on functionality offered by services, but these are not further considered here.

Combining Advertisements
Merging advertisements can be described as determining the *cross product* of the individual advertisements. Each advertisement alternative is combined with each other alternative, until all permutations are exhausted. Not all combinations of services are possible: The mediator agent uses domain-specific knowledge and combination policies to determine whether services can be offered together, and which should be offered as separate options.

Combining advertisements consists of two main tasks, described below in more detail: (i) Combining *service description terms*, and (ii) combining the *creation constraints* related to the service description terms. *Combining Service Description Terms*

```
<!-- Advertisement A1 from service provider agent P1 -->
<Terms>
 <ExactlyOne>
  <All>
   <ServiceDescriptionTerm serviceName="serviceA">
    <!-- domain-specific service description -->
   </ServiceDescriptionTerm>
   <ServiceDescriptionTerm serviceName="serviceB">
    <!-- domain-specific service description -->
   </ServiceDescriptionTerm>
  </All>
  <All>
   <ServiceDescriptionTerm serviceName="serviceC">
    <!-- domain-specific service description -->
   </ServiceDescriptionTerm>
  </All>
 <ExactlyOne>
</Terms>

<!-- Advertisement A2 from service provider agent P2 -->
<Terms>
 <All>
  <ServiceDescriptionTerm serviceName="serviceD">
    <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
 </All>
</Terms>
```

Example 3.6: Example advertisements.

When combining service description terms, a distinction between *combinable* and *non-combinable* services is made: Service descriptions are *combinable* when multiple services of the same type can be merged into one single service description. For example, on the power grid, electricity is combinable: service descriptions in advertisements from different utility providers can be combined into one advertisement. Service descriptions are considered *non-combinable* when services of the same type cannot be combined into one service description. For example, a CPU on one host cannot be easily combined with a CPU on another host on the fly, to provide an application with a single, virtual CPU.

The WS-Agreement service description term language does not provide the means to express whether services described within the terms are combinable or not. Therefore this knowledge has to be represented within domain-specific policies.

In the framework, it is assumed that a naming scheme is available for services: Different service description terms relate to the same service when their *serviceName* attributes have the same value. In domains where no common and unequivocal ontology for describing services is available, ontology mapping mechanisms should be applied to map different service descriptions onto a single service description ontology, which can then be used in the negotiation process.

To clarify the combination process, two examples are given. First, an example of the combination process involving several different services is presented. Next, an example is given in which two advertisements that both offer the same service are combined. Example 3.6 shows the service description terms of two example advertisements. A1 specifies two options: services {*serviceA, serviceB*} *or* {*serviceC*} may be requested. A2 specifies a single service that may be requested, service {*serviceD*}.

```
<ExactlyOne>
 <All>
  <!-- option from advertisement A1 -->
  <ServiceDescriptionTerm serviceName="serviceA">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
  <ServiceDescriptionTerm serviceName="serviceB">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
 </All>
 <All>
  <!-- option from advertisement A1 -->
  <ServiceDescriptionTerm serviceName="serviceC">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
 </All>
 <All>
  <!-- option from advertisement A2 -->
  <ServiceDescriptionTerm serviceName="serviceD">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
 </All>
 <All>
  <!-- option combining A1 and A2 -->
  <ServiceDescriptionTerm serviceName="serviceA">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
  <ServiceDescriptionTerm serviceName="serviceB">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
  <ServiceDescriptionTerm serviceName="serviceD">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
 </All>
</ExactlyOne>
```

Example 3.7: The combined advertisement.

The mediator agent first determines all possible combination options from both advertisements. This results in the following options: {*serviceA, serviceB*}, {*serviceC*}, {*serviceD*} (i.e. the original options specified in each of the advertisements), {*serviceA, serviceB, serviceD*}, and {*serviceC, serviceD*} (i.e. the options resulting from combining

both advertisements). Next, the mediator reviews the combinations for any combinations that are either not allowed or not possible. In this example, the option {*serviceC, serviceD*} is not allowed. This results in the option being removed from the advertisement. Example 3.7 shows the final advertisement.

In realistic situations, with more than two provider agents, each offering multiple services, the number of options in the merged advertisements can grow considerably. The normal form representation, however, ensures that the advertisement consists of a list of mutually exclusive options, which may become long, but has a nesting depth of 1. For readability, normal form documents can always be processed into a form which combines service options in a more compact, nested composition hierarchy. Example 3.8 shows a possible alternative representation of the advertisement in Example 3.7.

```
<ExactlyOne>
 <OneOrMore>
  <All>
   <ServiceDescriptionTerm serviceName="serviceA">
    <!-- domain-specific service description -->
   </ServiceDescriptionTerm>
   <ServiceDescriptionTerm serviceName="serviceB">
    <!-- domain-specific service description -->
   </ServiceDescriptionTerm>
  </All>
  <All>
   <ServiceDescriptionTerm serviceName="serviceD">
    <!-- domain-specific service description -->
   </ServiceDescriptionTerm>
  </All>
 </OneOrMore>
 <All>
  <ServiceDescriptionTerm serviceName="serviceC">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
 </All>
</ExactlyOne>
```

Example 3.8: Alternative representation of the combined advertisement depicted in Example 3.7, using nested term compositors.

When two or more service provider agents offer the same services, the service description terms describing these services can be combined into a single service description term, if these services are *combinable*. This combined term then represents a virtual combination of the underlying services. If the service described in the service description terms is *non-combinable*, the terms are treated separately, and combined as in the previous example.

The process of combining two service description terms describing the same service entails combining the domain-specific service description contained within the service description term. Example 3.9 shows an example, in which two advertisements offer the same service types: At the level of the service description terms, the advertisements have identical contents: They offer serviceA and serviceB. In this example, serviceA is considered to be a *combinable* service, while serviceB is a *non-combinable* service.

```
<!-- Advertisement A1 -->
<ExactlyOne>
 <All>
  <ServiceDescriptionTerm serviceName="serviceA">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
 </All>
 <All>
  <ServiceDescriptionTerm serviceName="serviceB">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
 </All>
</ExactlyOne>

<!-- Advertisement A2 -->
<ExactlyOne>
 <All>
  <ServiceDescriptionTerm serviceName="serviceA">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
 </All>
 <All>
  <ServiceDescriptionTerm serviceName="serviceB">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
 </All>
</ExactlyOne>
```

Example 3.9: Two example advertisements containing the same services.

```
<!-- Combined Advertisement A1 & A2 -->
<ExactlyOne>
 <All>
  <ServiceDescriptionTerm serviceName="serviceA">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
 </All>
 <All>
  <ServiceDescriptionTerm serviceName="serviceB">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
 </All>
 <All>
  <ServiceDescriptionTerm serviceName="serviceB">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
 </All>
</ExactlyOne>
```

Example 3.10: Combined advertisement based on the advertisements depicted in Figure 3.9.

The combined advertisement created from these advertisements by the mediator agent is shown in Example 3.10. When a service is known to be a combinable service, in this case serviceA, the service description terms can be combined: serviceA is reduced to a single service description term. serviceB is known to be not combinable, and so both service description terms are added to the combined advertisement.

Whether service description terms of *combinable* services can be combined or not, is also influenced by term compositors. Example 3.11 shows advertisements similar to those shown in the previous example. The advertisements differ, however, with respect to the term compositors. In this example, the services offered can only be requested as a pair: The All compositor indicates that they cannot be separated.

```
<!-- Advertisement A1 -->
<ExactlyOne>
 <All>
  <ServiceDescriptionTerm serviceName="serviceA">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
  <ServiceDescriptionTerm serviceName="serviceB">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
 </All>
</ExactlyOne>

<!-- Advertisement A2 -->
<ExactlyOne>
 <All>
  <ServiceDescriptionTerm serviceName="serviceA">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
  <ServiceDescriptionTerm serviceName="serviceB">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
 </All>
</ExactlyOne>
```

Example 3.11: Example advertisements containing paired options consisting of both *combinable* and *non-combinable* services.

Example 3.12 shows the combined advertisement, which simply contains the two underlying service provider agent advertisements as separate options, as the All compositor prevents serviceA from being treated separately from serviceB.

```
<!-- Combined Advertisement A1 & A2 -->
<ExactlyOne>
 <All>
  <ServiceDescriptionTerm serviceName="serviceA">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
  <ServiceDescriptionTerm serviceName="serviceB">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
 </All>
 <All>
  <ServiceDescriptionTerm serviceName="serviceA">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
  <ServiceDescriptionTerm serviceName="serviceB">
   <!-- domain-specific service description -->
  </ServiceDescriptionTerm>
 </All>
</ExactlyOne>
```

Example 3.12: Combined advertisement resulting from the combination of advertisements in Example 3.11.

Combining Creation Constraints
Not only do service descriptions need to be combined, negotiation constraints (i.e. the *creation constraints* in the advertisements) defined over these services must also be combined. Whether this is possible depends on the type of service, the restriction model used within the creation constraints, and on the policies governing the combination process. The restrictions as defined in our model are merged in the following way:

Maximum value restrictions:
The maximum value of all restrictions is the new maximum value.

Minimum value restrictions:
The minimum value of all restrictions is the new minimum value

Enumeration restrictions:
The union of all enumerated values is the new enumeration restriction.

```
<Template>
 <Name>Advertisement˙1</Name>
 <Context>...</Context>
 <Terms>
  <ServiceDescriptionTerm serviceName="serviceA">
   <Service><Amount/></Service>
  </ServiceDescriptionTerm>
 </Terms>
 <CreationConstraints>
  <Item>
   <Location>
    //ServiceDescriptionTerm[@serviceName='serviceA']/Amount
   </Location>
   <Restriction type="maxValue">
    <maxValue>400</maxValue>
   </Restriction>
  </Item>
 </CreationConstraints>
</Template>

<Template>
 <Name>Advertisement˙2</Name>
 <Context>...</Context>
 <Terms>
  <ServiceDescriptionTerm serviceName="serviceA">
   <Service><Amount/></Service>
  </ServiceDescriptionTerm>
 </Terms>
 <CreationConstraints>
  <Item>
   <Location>
    //ServiceDescriptionTerm[@serviceName='serviceA']/Amount
   </Location>
   <Restriction type="maxValue">
    <maxValue>250</maxValue>
   </Restriction>
  </Item>
 </CreationConstraints>
</Template>
```

Example 3.13: Two advertisements containing service descriptions for the same service and related constraints.

Whether combining creation constraints is meaningful, depends on the domain in which the advertisements are used. Example 3.13 shows two advertisements: *Advertisement_1* specifies the service *serviceA*, with a restriction indicating that the amount requested for this service may not exceed 400. *Advertisement_2* also specifies service *serviceA*, but specifies a restriction indicating that the amount requested may not exceed 250.

```
<Template>
 <Name>Combined_Advertisement</Name>
 <Context>...</Context>
 <Terms>
  <ServiceDescriptionTerm serviceName="serviceA">
   <Service>
    <Amount/>
   </Service>
  </ServiceDescriptionTerm>
 </Terms>
 <CreationConstraints>
  <Item>
   <Location>
    //ServiceDescriptionTerm[@serviceName='serviceA']/Service/Amount
   </Location>
   <Restriction type="maxValue">
    <maxValue>400</maxValue> <!-- value is MAX(250,400) -->
   </Restriction>
  </Item>
 </CreationConstraints>
</Template>
```

Example 3.14: Combination of advertisements in Example 3.13.

Example 3.14 shows how these two advertisements are combined: combining service description terms and creation constraints. Both service description terms are combined, as they both represent the same service. Furthermore, the creation constraints are combined: The maximum value of the two creation constraints is taken as the new maximum value, as this represents the maximum value that can be requested across both services.

The combined advertisements as shown in these examples are sent to the consumer agents: Upon receiving a request for advertisements from a consumer agent, the mediator agent examines the service interests contained within the request, and returns any advertisements that contain one or more services for which the consumer agent has expressed its interest.

Advertisement Phase: Service Provider Agent

For service provider agents, the advertisement phase entails (i) maintaining advertisements, and (ii) responding to requests for advertisements by the mediator agent of the virtual organization. Advertisements are an *indication* of the valid negotiation space, and that service provider agents may deviate from the values specified in the advertisements during negotiation, if the situation requires. For example, if a service suddenly suffers a drop in performance, a service provider agent may no longer be able to offer the service as specified in the advertisement. A service provider agent must ensure that the advertisements returned to the mediator agent reflect the current state of the services. To this end, a service provider agent must maintain an accurate view of available services, and lo-

cal service access policies, and translate these into service description terms and creation constraints.

```
<Terms>
 <OneOrMore>
  <All>
   <ServiceDescriptionTerm serviceName="serviceA">
    <!-- domain-specific service description -->
   </ServiceDescriptionTerm>
   <ServiceDescriptionTerm serviceName="serviceB">
    <!-- domain-specific service description -->
   </ServiceDescriptionTerm>
  </All>
  <ExactlyOne>
   <ServiceDescriptionTerm serviceName="serviceC">
    <!-- domain-specific service description -->
   </ServiceDescriptionTerm>
   <ServiceDescriptionTerm serviceName="serviceD">
    <!-- domain-specific service description -->
   </ServiceDescriptionTerm>
   <ServiceDescriptionTerm serviceName="serviceE">
    <!-- domain-specific service description -->
   </ServiceDescriptionTerm>
  </ExactlyOne>
 </OneOrMore>
</Terms>
```

Example 3.15: Term composition examples.

Specifying Creation Constraints

For each service description term specified in an advertisement, a service provider agent can define constraints that specify the valid value range for any service requests based on the advertisement. Creation constraints, and the restriction model as discussed in Section 3.3 are used for this purpose. Example 3.16 shows an example advertisement in which a constraint is defined for the *NegotiationIssue* element of a service with service type *serviceA*.

```
<Template>
 <Name>...</Name>
 <Context>...</Context>
 <Terms>
  <ServiceDescriptionTerm serviceName="serviceA">
   <ExampleService>
    <NegotiationIssue/>
   </ExampleService>
  </ServiceDescriptionTerm>
 </Terms>
 <CreationConstraints>
  <Item>
   <Location>
    //ServiceDescriptionTerm[@serviceName='serviceA']
                  /ExampleService/NegotiationIssue
   </Location>
   <Restriction type="maxValue">
    <maxValue>70</maxValue>
   </Restriction>
  </Item>
 </CreationConstraints>
</Template>
```

Example 3.16: An example service provider agent advertisement.

Service Combination Restrictions
In addition to creation constraints indicating limitations on the use of individual services, a service provider agent also specifies restrictions on the combination of individual services. Using the term compositors as specified in Section 3.3.1, a service provider agent indicates combination options. As an example, consider the terms shown in Example 3.15. In this example, the term compositors specify that either {*serviceA,serviceB*}, or one of the services {*serviceC, serviceD, and serviceE*} can be requested, or both (indicated by the *OneOrMore* compositor).

3.4.2 Request Phase

In the request phase, consumer agents select an advertisement from the advertisements obtained from the mediator agents, and use this to create a negotiation request. The request describes which services a consumer agent wishes to access, under which conditions, in accordance with the advertisement on which the request is based. The mediator agent translates the agreement request into one or more requests to the individual service provider agents involved.

Request Phase: Consumer Agent

If one of the advertisements the consumer agent has received is in line with its needs, the consumer agent formulates a negotiation request based on the advertisement, and communicates this to the mediator agent. If no appropriate advertisement is available, the consumer agent has two options: (i) change its requirements, or (ii) contact another mediator agent. In Example 3.17, the service description terms from the integrated advertisement (shown in Example 3.7) have been 'filled in' by the consumer agent to request specific values of the services described in the advertisement. In this case, the agent has based its request on the last of the three options presented in the advertisement.

Request Phase: Mediator Agent

In the request phase, the mediator agent is responsible for handling incoming requests from consumer agents, and translating these into requests for services with appropriate service provider agents.

One of the main tasks of a mediator agent is to allow consumer agents to request multiple services at the same time. The mediator agent has the responsibility of allocating the requested services to the service provider agents as efficiently as possible.

To this purpose, each incoming request from a consumer agent is intersected[1] with the advertisements of the provider agent, to determine which (combination of) service provider agent advertisements satisfies the consumer agent request. As an example, consider the the request shown in Example 3.17: The request is intersected with the advertisements provided by service provider agents P1 and P2 (see Example 3.6). As P1 provides services serviceA and serviceB, intersection with the advertisement of P1 results in services serviceA and serviceB. Similarly, intersection with the advertisement of P2 results in

[1] The *Intersect* operation defined in WS-Policy is used for this purpose.

```
<Agreement>
 <Name>...</Name>
 <Context>...</Context>
 <Terms>
  <ExactlyOne>
   <All>
    <ServiceDescriptionTerm serviceName="serviceA">
     <Service>
      <Amount>250</Amount>
     </Service>
    </ServiceDescriptionTerm>
    <ServiceDescriptionTerm serviceName="serviceB">
     <Service>
      <Amount>400</Amount>
     </Service>
    </ServiceDescriptionTerm>
    <ServiceDescriptionTerm serviceName="serviceD">
     <Service>
      <Amount>50</Amount>
     </Service>
    </ServiceDescriptionTerm>
   </All>
  </ExactlyOne>
 </Terms>
</Agreement>
```

Example 3.17: A consumer agent request.

service $\mathrm{serviceD}$. The resulting intersections are used by the mediator agent to generate requests to the service provider agents. Example 3.18 shows the resulting requests.

If several service provider agents offer similar services, the intersection process results in several request options, between which the mediator agent must choose. For example, if in the previous example $\mathrm{P1}$ had also offered $\mathrm{serviceC}$, the mediator agent would have had two alternative negotiation options from which to choose: either request all services from $\mathrm{P1}$, or to split the request into requests to $\mathrm{P1}$ and $\mathrm{P2}$ as described above. This selection process is governed by negotiation policies, specifying which combinations of providers are allowed or not, and which combinations are preferred over others. For example, a mediator agent may prefer to negotiate with as few service provider agents as possible. This policy would result in a mediator agent having to choose the option that contains the fewest separate negotiation requests.

Request Phase: Service Provider Agent

The request phase from a service provider agent's point-of-view consists of processing incoming negotiation requests from the mediator agent of the virtual organization. For each negotiation request, it determines whether it can provide the services advertised to the consumer agent in question, on the basis of local policies.

3.4.3 Offer Phase

In the offer phase, the mediator agent compares and combines service provider offers, and returns negotiation offers to consumer agents. Service provider agents analyze the negotiation requests they received and generate appropriate offers in response.

```
<!-- request to provider P1 -->
<Agreement>
 <Name/>
 <Context/>
 <Terms>
  <All>
   <ServiceDescriptionTerm servicename="serviceA">
    <Service>
     <Amount>250</Amount>
    </Service>
   </ServiceDescriptionTerm>
   <ServiceDescriptionTerm serviceName="serviceB">
    <Service>
     <Amount>400</Amount>
    </Service>
   </ServiceDescriptionTerm>
  </All>
 </Terms>
</Agreement>

<!-- request to provider P2 -->
<Agreement>
 <Name/>
 <Context/>
 <Terms>
  <All>
   <ServiceDescriptionTerm serviceName="serviceD">
    <Service>
     <Amount>50</Amount>
    </Service>
   </ServiceDescriptionTerm>
  </All>
 </Terms>
</Agreement>
```

Example 3.18: Service provider agent requests.

Offer Phase: Consumer Agent

In the offer phase, consumer agents receive negotiation offers from one or more mediator agents, and select the offer with the highest utility. To this end, a consumer agent compares the terms described in each of the offers from the mediator agents it has contacted, and determines which of the offers provides the highest utility for the agent. Each negotiation offer contains one or more options, from which the consumer agent must choose. Figure 3.8 shows an example in which a consumer agent must select between three offers, each containing a number of options. Offer A contains 3 options, offer B one option, and offer C two options. In the example, the consumer agent select *option 2* from offer *A*.

Offer Phase: Mediator Agent

In the offer phase, the mediator agent gathers all offers from individual service provider agents, and determines the possible allocations of services from these offers. The possible combinations are generated using the *merge* operation also used in the advertisement phase: The negotiation offers are translated to *normal forms*, after which the service description terms in the individual service provider agent offers are combined into a negotiation offer intended for the requesting consumer agent. This selection process consists of (i) comparing the utilities of the offers and (ii) applying organizational policies that

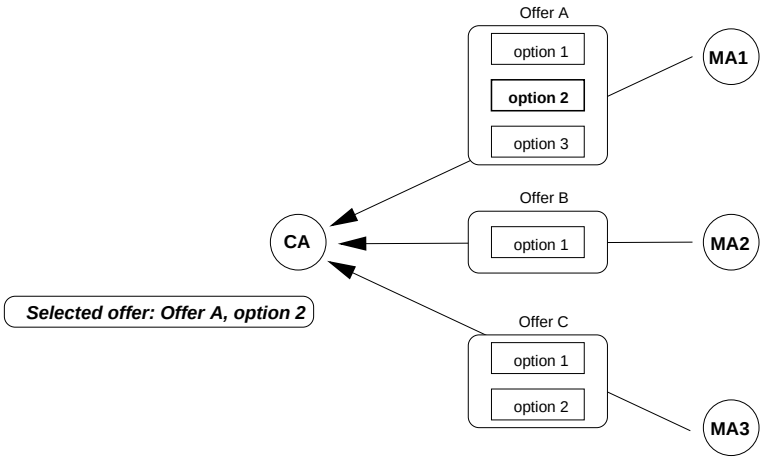

Figure 3.8: A consumer agent receiving 3 offers containing several options.

describe which combinations are allowed, which are not, and which combinations are preferred above other combinations, if the utility of these offers is equal. For example, a mediator agent may have the option to select between two offers containing the same services, one consisting of a combination of *two* service provider agent offers, and one consisting of a single provider agent offer. A possible policy which can be applied in this case is to select the offer with the lowest number of provider offers. Figure 3.9 depicts this example.

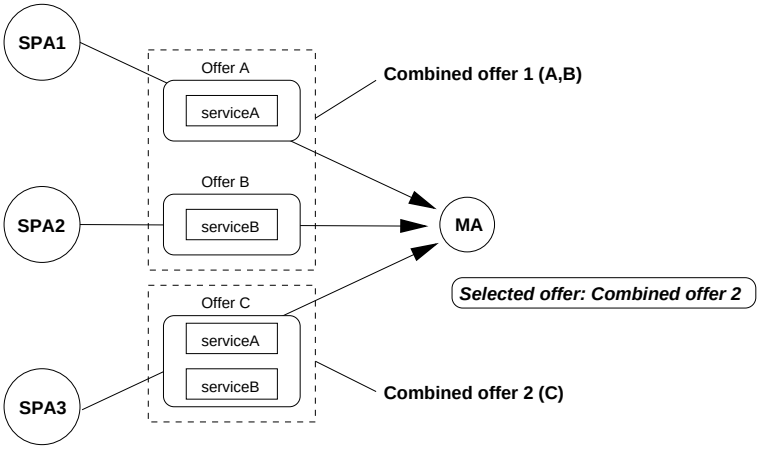

Figure 3.9: A mediator agent selecting between two offer combinations with equal utility. The offer involving the lowest number of service provider agents is selected.

Combining Service Description Terms

Creating offer combinations entails combining the service description terms contained within each offer. This process is similar to combining service description terms as described in the advertisement phase. Combining similar service description terms in this

phase however, is based on domain-specific knowledge. This domain specific knowledge is expressed in *combination rules*. The combination rules are specified using XML and XPath [14] expressions, and are used to describe how two instances of a similar service description can be combined into one. XPath allows for easy querying of XML specifications, and has basic logical support. XPath expressions can be used to select specific parts of XML document structures, manipulate the obtained information, and output the information in other XML document structures. In our framework, each combination rule uses an XPath expression to describe the elements of the service description term to which the rule applies, and the operations to be performed.

```
<CombinationRule serviceName="serviceA">
 <Service>
  <Amount>
   <!-- use the '+' operator to add the two amounts -->
   /Offer1/ServiceDescriptionTerm[@serviceName='serviceA']/Service/Amount +
   /Offer2/ServiceDescriptionTerm[@serviceName='serviceA']/Service/Amount
  </Amount>
 </Service>
</CombinationRule>
```

Example 3.19: Service description term combination rule.

The combination rule shown in Example 3.19 specifies how two instances of a term describing serviceA can be combined into a single service description term. The combination rule consists of a domain-specific service specification, containing XPath expressions to indicate how a combined value in the service specification is to be generated from existing values, and where the result should be placed in the combined service description term. In the example, the values of the *Amount* elements of both offers are added using the '+' operator.

As an example of the application of this rule, consider two similar service description terms in Example 3.20. Both services are of the same type: serviceA, which is a combinable service. The combination rule above applies, and results in a combined service description term, as shown in Example 3.21.

```
<!-- Term in Offer 1-->
<ServiceDescriptionTerm serviceName="serviceA">
 <Service>
  <Amount>250</Amount>
 </Service>
</ServiceDescriptionTerm>

<!-- Term in Offer 2-->
<ServiceDescriptionTerm serviceName="serviceA">
 <Service>
  <Amount>450</Amount>
 </Service>
</ServiceDescriptionTerm>
```

Example 3.20: Combining two service offers.

The above example shows the application of a simple combination rule involving a single attribute. The XPath expressions, however, can be used to define more complex combination rules, for example involving conditional elements and iterations over value sets.

```
<!-- The combined term after application of the rule -->
<ServiceDescriptionTerm serviceName="serviceA">
 <Service>
  <Amount>700</Amount> <!-- 250 + 450 -->
 </Service>
</ServiceDescriptionTerm>
```

Example 3.21: Result of combining the two offers in Example 3.20.

After the mediator agent has generated the offer combinations, the total utility of each of the offer combinations is calculated. The combination with the highest utility is selected (according to some domain-specific utility function), and returned to the consumer agent. Also, offers from service provider agents that have not been included in the offer to the consumer agent are explicitly rejected. Note that it is not required for a mediator agent to always return a single offer to the consumer agent: Multiple offers may have the same utility, or an offer selection policy may be implemented that explicitly requires multiple offers to be returned to consumer agents. Choosing between these offers is then left up to the consumer agents.

```
<Agreement>
 <Name>...</Name>
 <Context>...</Context>
 <Terms>
  <ExactlyOne>
   <All>
    <ServiceDescriptionTerm servicename="serviceA">
     <Service>
      <Amount>40</Amount>
     </Service>
    </ServiceDescriptionTerm>
   <All>
   <All>
    <ServiceDescriptionTerm serviceName="serviceB">
     <Service>
      <Amount>20</Amount>
     </Service>
    </ServiceDescriptionTerm>
   </All>
   <All>
    <ServiceDescriptionTerm serviceName="serviceA">
     <Service>
      <Amount>40<Amount>
     </Service>
    </ServiceDescriptionTerm>
    <ServiceDescriptionTerm serviceName="serviceB">
     <Service>
      <Amount>20</Amount>
     </Service>
    </ServiceDescriptionTerm>
   </All>
  </ExactlyOne>
 </Terms>
</Agreement>
```

Example 3.22: Example incoming negotiation request.

Offer Phase: Service Provider Agent

In the offer phase, a service provider agent generates negotiation offers for each of the negotiation requests it receives from the mediator agent.

Each negotiation request consists of one or more service description terms, describing the requested services. For each term, it determines whether the requested service can be offered, or not. The current state of the requested service is examined, and policies specified by the service provider are applied, to determine an appropriate offer in response to the request. If the request cannot be fulfilled, an alternative offer may be generated (i.e. the requested service can be partially delivered). For example, consider the negotiation request shown in Example 3.22. In this request, two services are requested: $\mathrm{serviceA}$ and $\mathrm{serviceB}$. The request specifies that offers are requested for either $\mathrm{serviceA}$ or $\mathrm{serviceB}$, or for $\{\mathrm{serviceA\ and\ serviceB}\}$.

In this example, the provider is capable of providing $\mathrm{serviceA}$ and $\mathrm{serviceB}$, but is more interested in delivering *both* services together, instead of individually. This preference is reflected by the provider in the negotiation offer, shown in Example 3.23: The amounts offered for each of the services individually are lower than the amounts offered for each of the services in the combined offer. For $\mathrm{serviceA}$ individually the amount offered is 30, instead of the requested 40. For $\mathrm{serviceB}$ individually the amount offered is 15 instead of the requested 20. Together however the requested amount of both $\mathrm{serviceA}$ and $\mathrm{serviceB}$ are included in the offer. Offering lower amounts for individual services lowers the likelihood of these offers being selected and accepted by the consumer.

3.4.4 Acceptance Phase

In the acceptance phase, agreement offers are accepted or rejected, and agreements are established and implemented.

Acceptance Phase: Consumer Agent

If the consumer agents agrees with (one of) the offers it has been made by a mediator agent, it sends that mediator agent an *accept* notification, containing the offer (and possibly which option contained within the offer, if the offer contained several options) that was accepted. In return, the consumer agent receives an agreement document, which it can use to claim the services specified in the accepted offer. If a consumer agent does not agree with an offer made by a mediator agent, it sends that mediator agent a *reject* notification, containing the offer that is being rejected.

Acceptance Phase: Mediator Agent

Mediator agents can either receive an *accept* or *reject* notification as a response to an offer made to a consumer agent. When a rejection is received, all service provider agents participating in the rejected offer are notified by the mediator of rejection of their offers. If an acceptance notification is received, the service provider agents that participated in the offer are informed of the acceptance. If an offer contained multiple options, the service

```
<Agreement>
 <Name>...</Name>
 <Context>...</Context>
 <Terms>
  <ExactlyOne>
   <All>
    <ServiceDescriptionTerm servicename="serviceA">
     <Service>
      <Amount>30</Amount> <!-- lowered to 30 -->
     </Service>
    </ServiceDescriptionTerm>
   </All>
   <All>
    <ServiceDescriptionTerm serviceName="serviceB">
     <Service>
      <Amount>15</Amount> <!-- lowered to 15 -->
     </Service>
    </ServiceDescriptionTerm>
   </All>
   <All>
    <ServiceDescriptionTerm serviceName="serviceA">
     <Service>
      <Amount>40<Amount> <!-- unchanged -->
     </Service>
    </ServiceDescriptionTerm>
    <ServiceDescriptionTerm serviceName="serviceB">
     <Service>
      <Amount>20</Amount> <!-- unchanged -->
     </Service>
    </ServiceDescriptionTerm>
   </All>
  </ExactlyOne>
 </Terms>
</Agreement>
```

Example 3.23: Negotiation offer response to request shown in Example 3.22.

provider agents that participated in the options that have not been selected are informed of rejection.

Each service provider agent of which an offer has been accepted returns an agreement document detailing the services it has agreed to provide. The mediator agent combines these agreements in one agreement document, specifying the overall services that will be delivered by the service provider agents. Finally, this combined agreement is returned to the consumer agent.

Acceptance Phase: Service Provider Agent

In the acceptance phase, service provider agents receive notifications of acceptance or rejection of the offer they have made. When an offer is rejected, the negotiation sequence is terminated, and all associated service operations (e.g. reservations) are rolled back. Each offer expires after a certain time-period specified by the service provider agent, to prevent unclaimed offers from consuming resources. When an offer (or offer option, if multiple options were offered during the offer phase) has been accepted, the provider agent implements the selected offer by ensuring the service is ready for use by the consumer agent. Finally, an agreement document is returned by each service provider agent to the mediator agent, detailing the services it has agreed to provide.

3.4.5 Negotiation Process Overview

Figure 3.10 shows a complete overview of a single negotiation sequence. For each of the negotiation phases, the main tasks of the participants are indicated. Arrows represent the exchange of negotiation documents between participants. In most negotiation situations, mediators will negotiate with multiple service providers concurrently. However, for simplicity, the figure only shows a single service provider participant. In the next section, the tasks indicated in the figure are discussed in more detail.

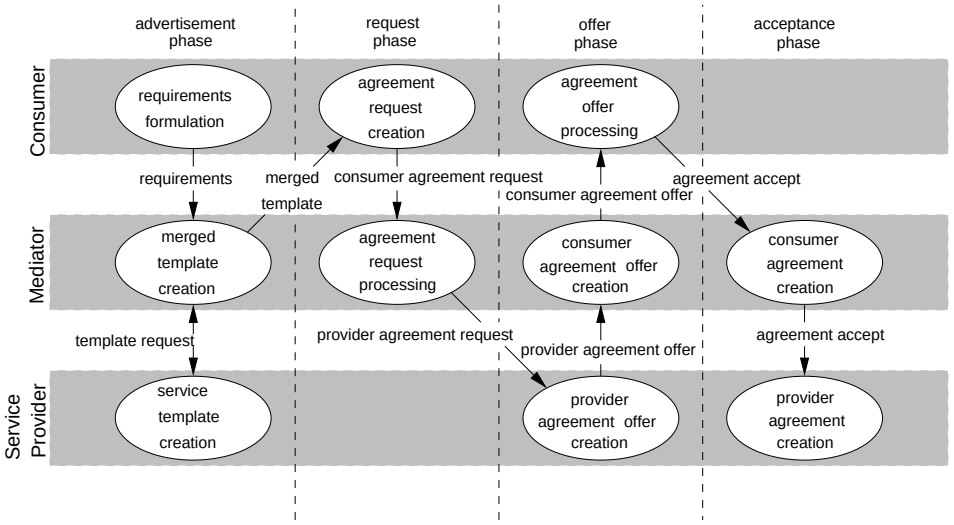

Figure 3.10: The negotiation process.

3.5 Agent Task Models

In this section, for each of the agents of the framework, the internal tasks and their relationships are identified, including the role of domain-specific knowledge in these tasks, and the information exchanged between the negotiation parties as a result of these tasks. The information exchange is described using *calls* performed on *interfaces*, but can also be regarded as messages exchanged between parties, or any other method of communication that is deemed appropriate in the application domain.

For each of the three agent types (consumer/mediator/service provider), the following three high-level tasks are identified:

1. ADVERTISEMENT MANAGEMENT
2. REQUEST MANAGEMENT
3. AGREEMENT MANAGEMENT

The tasks are related to the four phases of the negotiation process (see Figure 3.10): The ADVERTISEMENT MANAGEMENT task is responsible for performing tasks related to

the advertisement phase; the REQUEST MANAGEMENT task is responsible for handling tasks related to the request phase and the offer phase; the AGREEMENT MANAGEMENT task is responsible for handling tasks related to the acceptance phase.

For each of these tasks, subtasks are identified, and domain-specific information that is required is specified. Tasks and subtasks are indicated in the following figures using *solid boxes*, domain-specific information is indicated using *dashed boxes*. Dependencies between tasks are indicates using *dashed arrows*.

3.5.1 Consumer Agent Tasks

Consumer agents perform three main tasks related to the negotiation process: Maintaining mediator advertisements, creating negotiation requests, and selecting negotiation offers. Figure 3.11 gives an overview of the tasks required within the negotiation task of the consumer agent.

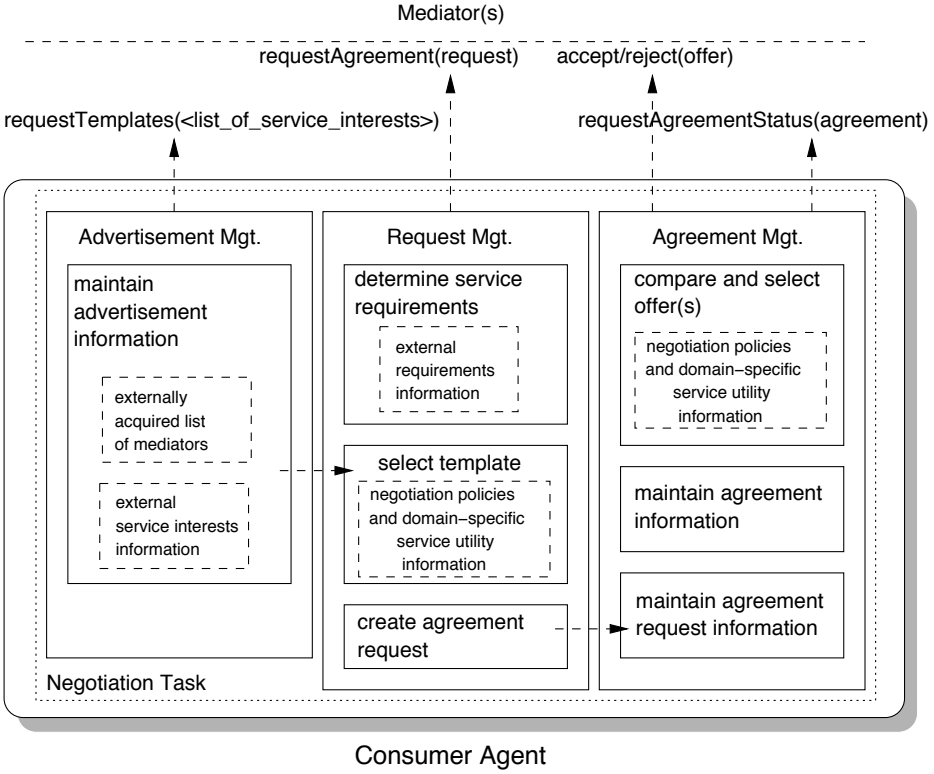

Figure 3.11: Tasks within the consumer agent.

1. Advertisement Management

The ADVERTISEMENT MANAGEMENT task is responsible for retrieving and storing advertisements. A consumer agent starts the negotiation process by requesting the avail-

able advertisements from mediator agents, using *requestTemplates('service interests list')* available on the interface of the mediator. A list of advertisements is returned by the mediator.

2. Request Management

The REQUEST MANAGEMENT task of the consumer agent is responsible for creating and sending out negotiation requests, and collecting offers in response to these requests.

The consumer agent selects an advertisement on which to base its negotiation request, governed by negotiation policies and domain-specific utility functions. The advertisement is used to create a negotiation request, which contains the service descriptions of the required services, and the desired service levels. The consumer agent must ensure that the request complies with the constraints specified within the advertisement, as the request will be denied by the mediator agent if this is not the case. The request is communicated to the mediator agent using the *requestAgreement(request_document)* call.

The consumer agent receives the replies to the negotiation request(s) from the mediator agents in the form of negotiation offer documents, and selects which negotiation offer to use. This selection process is governed by domain-specific policies, which use domain-specific service utility calculation mechanisms, to compare individual service offers within the negotiation offer documents.

3. Agreement Management

The AGREEMENT MANAGEMENT task of the consumer agent is responsible for maintaining established agreements. After an offer has been selected, it is accepted or rejected by notifying the mediator agent, using *accept(offer_document)*, or *reject(offer_document)*, with *offer_document* being the offer that is being accepted/rejected. As a result of acceptance, a final agreement document is returned by the mediator agent, on the basis of which the agent can claim the services. The claiming process is outside the scope of the negotiation framework, and this should be implemented at the service implementation level.

3.5.2 Mediator Agent Tasks

The mediator agent contains a negotiation module similar to the module of the service provider agents. The external negotiation interface is identical to the service provider agent interface. The tasks which need to be performed within the components of the module are however different. Figure 3.12 shows an overview of the tasks within a mediator agent, and the interactions with consumer and service provider agents.

1. Advertisement Management

The ADVERTISEMENT MANAGEMENT task of the mediator agent is responsible for collecting and maintaining the advertisements of the individual service providers within the virtual organization. Advertisements are requested from the service provider agents using the *requestTemplates()* call. The resulting advertisements are combined. Consumers can

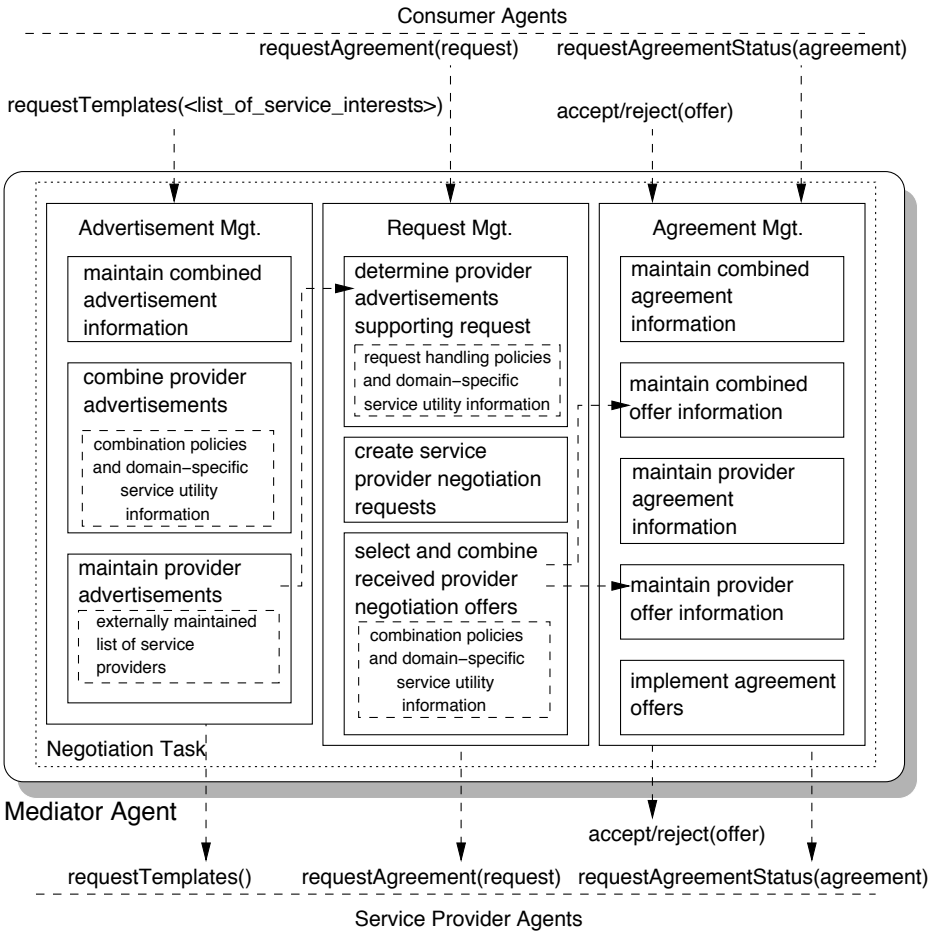

Figure 3.12: Negotiation tasks within the mediator agent.

request the available advertisements from a mediator using the call *requestTemplates('list of service interests')*.

2. Request Management

The subtasks of the REQUEST MANAGEMENT task are: (i) Determining which service provider agents can provide the services requested by a consumer, (ii) creating the negotiation requests to these service provider agents, and (iii) processing the individual provider offers into a combined offer for the consumer agent.

Consumer agents communicate negotiation requests to a mediator agent using *requestAgreement(request_document)*. The request document is translated into one or more negotiation requests to service provider agents by the mediator. This process is governed by domain specific organizational policies, which are implemented by this task. These requests are communicated using the *requestAgreement(request_document)* call.

The negotiation offer documents collected by service provider agents as a response to the negotiation request, are received by the mediator, and an offer selection process is performed, based on domain-specific utility calculation for the service levels offered in the individual offers, and governed by combination policies. The selected offer documents are then combined into a single offer and returned as such to the consumer agent. Finally, the combined offer, and the underlying offers made by the service provider agents are communicated to the Agreement Management component.

3. Agreement Management

The AGREEMENT MANAGEMENT task is responsible for maintaining information on four types of negotiation documents:

1. Information on negotiation offers made to consumer agents,
2. and their underlying negotiation offers made by the individual service provider agents;
3. Information on active agreements,
4. and their underlying service provider agent agreements.

Furthermore, agreement offers are implemented and monitored by this task. Upon receiving *accept(offer_document)* from a consumer agent, an agreement must be established, by accepting the underlying agreement offers made by the service provider agents. Similarly, when an agreement offer is rejected, the underlying offers from the service provider agents are also rejected.

Upon receiving *requestAgreementStatus(agreement_document)* from a consumer agent, the agreement management task returns status information on the requested agreement document. This information is obtained by combining the status information on the individual service provider agent agreements underlying the agreement.
The *requestAgreementStatus(agreement_document)* call is used to request status information from service provider agents.

Managing active agreements at the mediator level is a relatively simple task, as actual monitoring and enforcement of agreement is handled at the service provider agent level. The mediator agent is primarily responsible for monitoring the status of the service provider agreements underlying the currently active agreements, and take appropriate action if one or more of the underlying agreements expires or is violated. This process is governed by domain-specific organizational policies that are implemented by this task. (e.g. violation of a service provider agent agreement may not necessarily lead to expiration of an entire agreement that has been established with the consumer agent).

3.5.3 Service Provider Agent Tasks

A service provider agent provides negotiation facilities to allocate services to consumers, and is responsible for monitoring and controlling the services it offers. It is the responsibility of the service provider agent to translate service usage and access policies into advertisements and negotiation policies.

In addition the negotiation tasks which are also described for the consumer and mediator agents in Sections 3.5.1 and 3.5.2, a service provider agent also contains the SERVICE MANAGEMENT task, which provides monitoring and control facilities for the actual services provided. Figure 3.13 shows the tasks within the service provider agent.

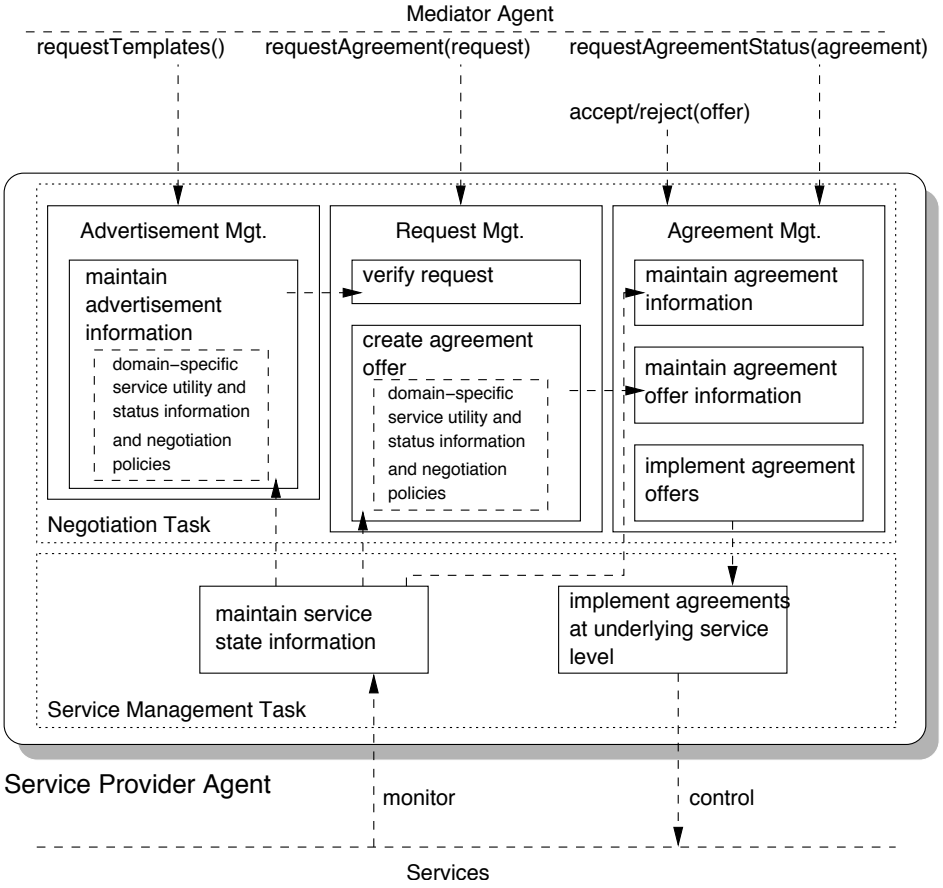

Figure 3.13: Negotiation tasks within the service provider agent.

1. Advertisement Management

The AGREEMENT MANAGEMENT task ensures that the advertisements reflect the status of the available services. Depending on the service domain, this may be implemented by periodic monitoring of services and updating of the advertisement documents. Service monitoring information can be obtained from the SERVICE MANAGEMENT task. Advertisement content is not only based on service status information alone, but also on domain-specific policies specifying which service capacity is provided to whom and when. For example, policies may prescribe limits on individual service requests based on administration level regulations.

2. Request Management

The REQUEST MANAGEMENT task consists of two main tasks: (i) Verifying incoming negotiation requests from mediators, and (ii) creating an appropriate response to requests. Requests are verified by comparing them to the advertisements on which they are based. When a request has been successfully verified, an offer may be made. An offer is based on the service provider agent's negotiation policies concerning the requested services, and the current status of the requested services. The status of a service can be monitored by requesting monitoring information from the SERVICE MANAGEMENT task.

3. Agreement Management

The agreement management task contains three main tasks: (i) Managing outstanding negotiation offers, (ii) implementing accepted offers into active agreements, and (iii) managing active agreements.

Maintaining negotiation offers entails (i) expiring offers which have exceeded the allowed offer duration, and (ii) ensuring that the services described in the offer are available for use if the offer is accepted (i.e. handling reservations). The expiration of a negotiation offer may involve releasing of reserved services, which can be performed by using the SERVICE MANAGEMENT task.

When a mediator agent accepts a negotiation offer, a final agreement document is created, and through the SERVICE MANAGEMENT task the specified services are made available to be claimed by consumer agents. Service reservations must be implemented and enforced at the service management level. Depending on the service domain and the capabilities of the SERVICE MANAGEMENT task in that domain, this enforcement may consist of controlling and/or monitoring of service usage. Rejection of a negotiation offer by the mediator agent results in the services being released.

Information on the current status of active agreements is maintained, (i) to respond to agreement status requests from the mediator agents, (ii) to handle any violations that may occur, and (iii) to handle agreement expirations. Service status information is collected from the SERVICE MANAGEMENT task, and used to determine if services are being delivered as specified in the agreement, or if violations have occurred (either by the consumer or the service provider agents).

Expiration of an agreement can occur as a result of a violation, or when the time-period agreed upon in the agreement has passed. The related services are released.

4. Service Management Task

The SERVICE MANAGEMENT task provides the glue between the domain-independent agreement negotiation functionality, and the actual services. Two main tasks are distinguished: (i) Maintaining service state information, and (ii) implementing service reservations (during negotiations) and agreements (after negotiations) at the underlying services.

The first task implements domain- and service-specific monitoring facilities, and translates this information into service state information to be used by the negotiation tasks to base it decisions upon. The second task implements service control facilities enabling reservation of services and implementation of established agreements. As this task is very

domain-specific, no further details are provided in the negotiation framework, other than the two tasks described above. An implementation of the service management task is discussed in Chapter 5.

3.6 Discussion

In this chapter, a framework for mediated negotiation between service consumers and service providers is presented. The framework consists of: (i) A two-tiered negotiation protocol and a domain-independent negotiation language; (ii) mechanisms for performing basic mediation operations on the negotiation documents; (iii) task models for the negotiation participants.

The negotiation protocol and language are based on the WS-Agreement specification. This specification is chosen as it presents one of the first attempts to standardize negotiation interactions in the area of service oriented computing. As our negotiation framework is intended for use in large-scale and heterogenous middleware environments, the use of standards is an important requirement. The basic WS-Agreement specification however provides a very limited negotiation model, which limits the possible negotiation interactions. For example, the specification allows only one-to-one interactions between service providers and consumers, and only a single negotiation round is supported. The WS-Agreement specification defines abstract agreement factories responsible for agreement negotiation and creation, which do not preclude the use of multiple agreement factories, to establish combined agreements. However, this is not further addressed in the specification.

Our framework extends the existing WS-Agreement specification by explicitly supporting mediation operations on the negotiation documents such as combining advertisements and negotiation offers, and the domain-specific service descriptions contained within. The WS-Agreement negotiation protocol is extended: The original *template* phase is extended to allow initiators of negotiations to express service interests. Based on these interests, specific advertisements can be returned, or advertisements can be created on the fly, tailored to the service interests of the initiator. Furthermore, the protocol is extended with an explicit acceptance/rejection phase at the end of the negotiation sequence.

The addition of the acceptance phase to the WS-Agreement protocol in our framework changes the semantics of the final WS-Agreement phase: The returned document in this phase is no longer considered a final agreement, but instead is considered an *offer*. This has implications with respect to the interoperability of our protocol with implementation of the WS-Agreement protocol by external parties.

In the case where external parties interoperate with an implementation of our framework, the implementation has to recognize that the final acceptance phase is not known to the other party, and accept the returned negotiation offers on behalf of the other party. In the case where an implementation of our framework needs to interoperate with other WS-Agreement based infrastructures, the implementation has to recognize that negotiation offers returned by the external party do not need to be explicitly accepted. This also implies that negotiations cannot be performed concurrently with multiple external parties, as de-commitment during negotiations is not possible.

IBM's Cremona [63] (Creation and Monitoring of Agreements) presents an architecture and set of libraries that implement the WS-Agreement interfaces and agreement (template) management, and provide agreement functionality suitable for implementations in domain-specific environments. The Cremona architecture specifies domain-independent and domain-specific components required for agreement-based management, and the Cremona libraries provide implementations of the agreement interfaces, domain-independent components, and well-defined interfaces for the domain-specific components. The Cremona architecture, however, only models direct consumer-provider interactions, and does not explicitly model the role of a mediator in the agreement negotiation process. Although the architecture could be used as a basis for mediator based negotiations, the absence of the final acceptance phase in the WS-Agreement protocol, as described earlier, would prevent a mediator from engaging in multiple negotiations simultaneously.

Independent from the WS-Agreement Specification activities, Hung [47] proposes a Web service negotiation model called WS-Negotiation, and presents a service level agreement (SLA) template model, with different domain specific vocabularies for supporting different types of negotiation. The negotiation protocol in their model is geared toward integrative negotiation, where both parties locate and adopt the option that provide greater joint utility to the parties taken collectively. The message types resemble those in our negotiation model and is more extended than the models presented by Paurobally and Jennings [74] and the GRAAP working group [9].

Extending the negotiation protocol as in our framework gives it more flexibility, making it possible to specify more elaborate negotiation models. In its basic form, our negotiation protocol can be compared to the well-known Contract Net protocol [87]: At both the consumer level and the provider level of the negotiation model, a Contract Net-like interaction sequence takes place: negotiation requests can be compared to *calls for proposals*, negotiation offers can be compared to *bids*, after which offers can be accepted or rejected, similar to acceptance/rejection of bids in the Contract Net protocol.

The extension of the negotiation protocol, as well as the explicit two-layered mediated negotiation model, allows our framework to implement a range of negotiation models on top of the WS-Agreement base.

For example, Sandholm and Lesser [84] describe an extension of the Contract Net protocol, in which de-commitment penalties are included in the contract negotiation process. Ending a contract negotiation results in the responsible party paying the penalty to the other party. Extending our model to include de-commitment penalties is possible: The negotiation documents used in the negotiation process can be extended to express de-commitment penalties, in addition to the other negotiation terms described earlier.

Furthermore, the mediator agent in our framework allows negotiations to not be limited to one-to-one interactions, but to include multiple negotiation participants represented by mediator agents. Also the mediator agent, as it is situated between the negotiation participants, can decouple negotiation interactions on both negotiation levels, to implement different negotiation models. For example, multiple negotiation rounds can be performed on one level, while limiting the negotiation interactions to a single round with the negotiation participants on the other level.

In the following chapter, the flexibility of the negotiation framework is demonstrated. The basic negotiation model is instantiated and demonstrated in the domain of distributed

energy management. Subsequently, scenarios are discussed in which the negotiation framework is adapted to (i) enable the use of de-commitment penalties, and (ii) to support consumer competition, through implementing an auction-like interaction model using the framework, in which the mediator agent takes on the role of auctioneer.

Chapter 4

Distributed Energy Management: Mediated Service Negotiation in Energy Markets

4.1 Introduction

This chapter describes the application of the negotiation framework in the domain of *Distributed Energy Management* in open energy markets. In these newly emerging markets, traditional energy management systems do not suffice: Distributed management infrastructures are required to facilitate interactions between market participants. The framework described in this thesis is used as a basis for an energy negotiation model, allowing energy providers to establish agreements concerning the supply of energy to consumers.

In this chapter, different negotiation models suitable for the domain of energy management are implemented using the framework. First, the framework is instantiated and negotiation policies are defined to implement two forms of negotiation: (i) Mediated provider competition in which a single mediator acts on behalf of the consumer agents, and (ii) mediated provider competition with multiple mediator agents. Scenarios are presented and simulations are performed to evaluate the behavior of these two negotiation models.

After this, possibilities for extending the negotiation framework to accommodate for other negotiation models are examined. Two extensions of the framework are described: (i) The integration of leveled-commitment contracts, allowing negotiation participants to drop commitments in favor of other negotiation options during negotiations, and (ii) the support for consumer competition by implementing an auctioning model. Two scenarios are presented in which the required extensions to the framework are discussed.

This section introduces the domain of distributed energy management. Section 4.2 introduces the energy negotiation model. Section 4.3 presents two scenarios in which the negotiation framework is instantiated for the two different negotiation types. Section 4.4 discusses two scenarios in which extensions of the negotiation framework are discussed. Conclusions and contributions are discussed in Section 4.5.

4.1.1 Distributed Energy Management

The term Distributed Energy (or Distributed Generation) is used to describe a relatively new area of development in the field of power generation, in which small scale distributed energy resources are used to provide energy at or near the point of use.

Ackermann et al. [4] recognize the absence of a general definition for the concept of distributed generation, and analyze the differences in the definitions and issues used in literature. They define distributed generation as follows:

> Distributed generation is an electric power source connected directly to the distribution network or on the customer side of the meter.

They also define *distributed utility architectures* as architectures that include distributed generation, demand-side mechanisms for influencing energy demand (e.g. load management systems) and mechanisms for providing transmission and distribution capacity (e.g. stand-by generators).

The potential benefits of distributed utility architectures has lead to substantial interest of governmental organizations [3] and international energy consortia [2], leading to an increase in research on technologies for enabling distributed generation in emerging open energy markets. Kok et al. [54] describe five driving forces behind the growing interest in the field:

Environmental concerns: Due to governmental support, a growing number of small scale, distributed, sustainable energy resources (wind turbines, solar arrays) are appearing within the market.

Open energy markets: Deregulation of electricity markets has created opportunities for medium and small generation of electricity, requiring lower investments and having shorter payback periods.

Diversification of energy resources: To reduce dependence on fossil fuels, and to cope with the global increase in energy demand, alternative energy sources are being researched.

Energy autonomy: The use of local energy capacity to allow for stand-alone operation of buildings or subsystems during power failures.

Energy efficiency: Transporting energy results in power loss. By using generated electricity locally, the loss is reduced, as well as transportation costs. Furthermore, the use of combined heat-power (CHP) generation is increasing in consumer homes.

Interaction and contracting mechanisms in current energy markets are not well suited for these new and dynamic distributed energy markets: Current markets are oriented towards limited competition between a small number of large providers, each with a large and fairly static customer base.

As energy markets are moving towards more fine-grained and distributed infrastructures, novel management approaches are required to facilitate the interactions between large and dynamic populations of energy providers and energy consumers. Automated negotiation and coordination infrastructures can be useful in these circumstances. Below, a number of agent-based approaches are discussed.

4.1.2 Related Work

Distributed energy management can be modelled as a multi-agent system: Agents are used to model the relevant entities, and agent communication and coordination models are used to specify the interactions between these parties. Below, a number of approaches to energy management using multi-agent systems is discussed.

Some earlier work in the area of distributed energy management using multi-agent systems focuses on *load management*. The main purpose of load management is to reduce peak loads, thereby reducing energy production costs, and the risk of energy shortages.

Akkermans et al. [6] describe a load management system in which electrical appliances are represented by so-called Homebots. Homebots participate in a computational market and bid for energy in an auction, initiated by the utility. The auction scheme consists of three stages: *announce*, *bid*, and *award*. Multiple bidding rounds are performed, until a globally optimal allocation is found. The utility acts as an equal party to the Homebots in the auction process, by using *utility interface agents* to influence the auction process. These agents are controlled by the utility, and can be used to influence auctions, either by reducing the load consumed by the interface agent, or changing the utility function of the interface agent (effectively 'buying back' energy).

Another example of an agent-based load management system is described by Brazier et al. [18]. In this work, a multi-agent system is presented in which *Customer Agents* negotiate with a *Utility Agent*. Negotiations are initiated by the utility agent. Three negotiation interaction methods are available within the system. Agents are able to choose between these different negotiation interactions, resulting in different negotiation strategies:

1. *Offer*: A utility agent makes an offer to customer agents, who may only answer '*yes*' or '*no*'. Customer agents are not able to influence the negotiation process in any other way.

2. *Request for bids*: A utility agent issues a request for bids to customer agents. Customer agents respond by indicating the desired amount of electricity, and the price they are willing to pay for this amount. Based on the response, the utility agent determines if additional negotiation rounds are necessary.

3. *Announcing reward tables*: Similar to the second method, but the utility agent now also restricts the possible bids by issuing a reward table with the request, indicating allowed bids.

Depending on the circumstances, one strategy may be preferred over another. For example, the first strategy provides little negotiation freedom, but is relatively fast. The second strategy gives customer agents the freedom to express bids, which can lead to multiple negotiation rounds, increasing overall negotiation time. The third strategy can be seen as a 'middle ground' strategy: Customer agents are allowed to make bids, but only within the negotiation space indicated by the utility agent.

The framework has been implemented and tested in DESIRE [19]. The main motivation of this work is to provide an automated negotiation system for managing dynamic and open energy markets. The work does not explicitly model concepts such as distributed

generation or use of localized energy resources. More recently, research has emerged incorporating these concepts:

Kok et al. [54] describe *PowerMatcher*, an agent-based hierarchical structure of *Supply and Demand Matchers* (SDMs). In this system, each SDM is responsible for increasing the match of the electricity production and consumption of the cluster directly below. The 'leaf' SDMs are the 'Device Agents' representing actual consumers and producers of energy. Different device types are modeled with varying behavioral models and levels of 'controllability'. Device agents implement a self-interested bidding strategy, based on achieving economic gain, while adhering to the operational constraints of the device. The 'root' SDM defines which market-mechanisms are available, and is also responsible for the price forming process. At specific intervals defined in the market, the root SDM sends out a request for bids to the lower SDMs, which are propagated down the hierarchy. The bids are aggregated, and the root SDM determines the equilibrium price. Devices can subsequently determine their respective power allocations based on the established market price and their bid functions. This framework models the concept of localized energy usage well, by defining both 'consumer' and 'provider' agents as equal parties in the market ('device agents'), and clustering these devices in an SDM hierarchy. The auction-like interaction within the hierarchy has the drawback that interactions are synchronized 'globally', and have to be initiated by the root 'SDM', decreasing the autonomy of the SDMs at the lower levels.

The Australian CSIRO science agency also focuses on developing mechanisms to facilitate new interaction forms between distributed electricity providers, electricity networks, and end-users: In a recent paper, James et al. [48] present an agent-based framework providing loads and generators with a market-based interaction environment. In the framework, a number of agent roles are distinguished, such as *buyer*, *seller*, and *device* roles. Agents can assume more than one role, and interact through a bulletin board. The model incorporates aggregation and brokering of capacity of distributed energy resources. Aggregated capacity is modeled using four issues: quantity, time(liness), reliability, and traceability to a known set of customers. Furthermore, this work stresses the importance of 'any-time' solutions, to ensure that adequate solutions are available in a short time. In contrast to the other approaches described above, this last approach can be described as *bottom-up*: Buyer agents initiate negotiations by issuing a *request for price* to a seller agent. The use of decentralized clustering is proposed for the formation of groups of buyers and sellers.

In the context of the POWERACE[1] project, Veit et al. [94] introduce an agent-based simulation model for determining the effects of CO^2-emission trading, and the influence of renewable energy sources in liberalized energy markets. Market participants are modeled as autonomous agents: The simulation enables the evaluation of different and dynamic adoption of strategies of participants. The model explicitly incorporates prediction of fluctuations of renewable energies.

Lastly, the *Alternative Energy Systems Consulting*[2] (AESC) firm has developed agent-

[1] http://www.powerace.de
[2] http://www.aesc-inc.com

based systems in cooperation with the *Electric Power Research Institute* (EPRI) and the *California Energy Commission* (CEC): Agent-based systems implementing an electronic auction for buying and selling electric power, and for control and scheduling of distributed energy resources were created.

4.2 Energy Negotiation Model

This section introduces a negotiation model for distributed energy management, based on the negotiation framework presented in Chapter 3. The model supports: (i) Aggregation of energy capacity, allowing medium- and small-size energy providers within a virtual organization to combine their offered energy capacity; (ii) *negotiation* as the interaction model, giving the participants autonomy in their interactions with other parties; (iii) negotiations initiated by energy consumers, not synchronized by a central manager.

The architecture distinguishes itself from the aforementioned systems by two elements: Firstly, negotiation is based on *mediation*. Groups of energy providers are represented by *group managers*, that implement the role of mediator in the negotiation framework. Although other approaches recognize hierarchies in which providers can represent multiple, lower-tier providers, our framework explicitly defines a *mediator role*. Our mediator agents do not only act in the best interest of the overall negotiation process, but actively influence the negotiation process by applying organization-wide policies. Secondly, negotiation is based on *advertisements*. Energy providers specify energy negotiation conditions in advertisements These advertisements are combined and offered to consumers by mediator agents. Figure 4.1 shows the mapping of the negotiation framework to the elements of the distributed energy management architecture.

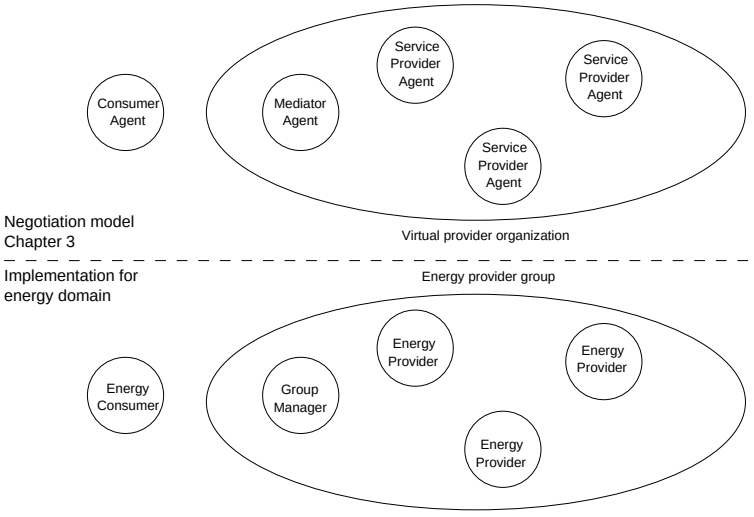

Figure 4.1: Overview of mediated energy negotiation model.

A *Group Manager* is introduced in the energy negotiation model, implementing the mediator agent role in the negotiation process. A Group manager can be either a sepa-

rate entity specifically created to implement the mediator agent role, or it can be an extended energy provider agent, implementing both the mediator and service provider agent roles. Energy providers and energy consumers negotiate with each other through group managers. The goal of negotiation is to establish agreements between consumers and providers about the delivery of energy resources. In the next section the negotiation participants, and the energy resource description used in the negotiation process to represent energy resources, are discussed

4.2.1 Consumer Agent

Energy consumer agents represent individual parties requiring access to a specific amount of energy. Figure 4.2 shows a typical consumer demand curve, specifying the demand over the course of one day.

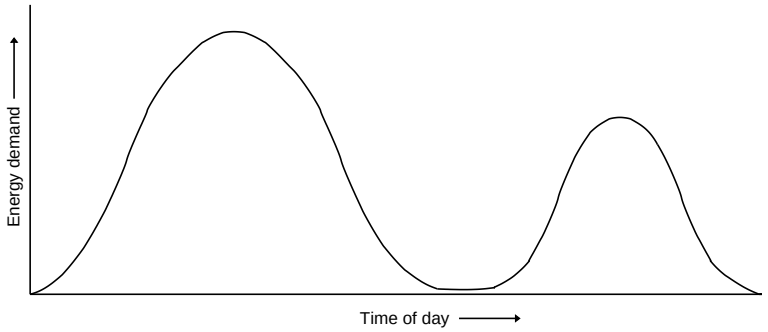

Figure 4.2: A typical consumer demand curve.

Consumer agents negotiate with one or more mediator agents representing energy providing organizations, to reach an agreement. An agreement guarantees access to energy resources for a certain time-period. In our model, the demand curve of an energy consumer agent is translated into a number of negotiation requests, aimed at obtaining the required energy for a certain time-period. Figure 4.3 depicts the translation of the demand curve.

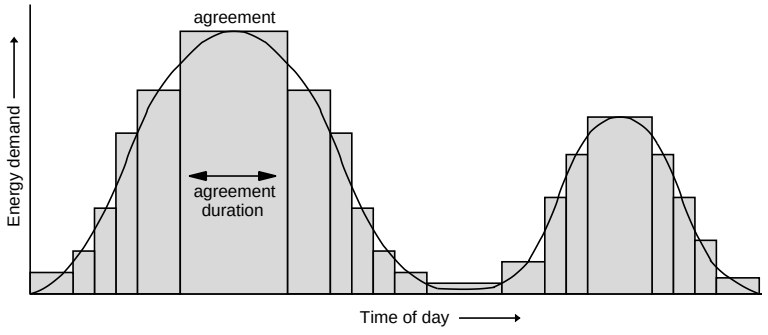

Figure 4.3: The demand curve translated into agreements.

Consumer agents initiate negotiations directly prior to the moment that energy is to be claimed and used: No advance negotiations for future energy demands are currently supported. Consumer agents negotiate for a single agreement covering the desired energy over a time-period: An agreement cannot be split across multiple mediator agents.

4.2.2 Mediator Agent

Energy provider agents are combined into groups. These groups can represent different types of real-world energy provider organizations, such as geographically based or administrative organizations. A mediator agent (i.e. energy provider group manager) represents a group in the negotiation process, and negotiates with individual *energy provider agents* on behalf of *consumer agents*. Decision making within mediator agents is based on two types of knowledge: (**A**) Knowledge concerning the types of energy resources in the organization, and (**B**) Negotiation policies. The mediator uses this knowledge at three points within the negotiation process:

1. To combine advertisements from energy provider agents (advertisement phase);
2. To select energy provider agents with whom to negotiate, when a negotiation request from a consumer agent is received (start of request phase);
3. To select and combine negotiation offers received from energy provider agents (end of request phase).

For each of the two knowledge types, its role at these points in the negotiation process is discussed below.

A: Mediator Agent 'Energy Resource' Knowledge

A mediator agent has knowledge related to the different types of resources provided by the providers within the organization. For example, an organization of a energy provider agents offering energy generated by *solar arrays*, and others offering energy generated by *diesel generators*. These resources essentially all offer the same basic resource (i.e. energy), but with different characteristics. Some resources may be aggregated without problems, but others may not fit well together. For example, when the capacity of a diesel generator is combined with the capacity of a wind generator, the overall reliability of the offered energy will be as low as the reliability of the wind generator. The mediator must take these aspects into account.

A1. Combining Advertisements

To combine provider advertisements, the mediator uses the knowledge it has on (i) the compatibility of different resources, and on (ii) how to combine advertisements containing descriptions of these resources.

A2. Selecting Energy Provider Agents

To handle an incoming negotiation request from a consumer agent, the mediator agent must decide which energy provider agents are suitable negotiation candidates, and start negotiations. For this purpose, the mediator agent must be aware which energy provider agents provide *similar* resources. These providers are candidates for negotiation. The selection process uses knowledge concerning the similarity of energy resources, and knowledge describing how consumer-level negotiation requests can be translated into one or more provider-level negotiation requests.

A3. Selecting Energy Provider Agent Negotiation Offers

Upon receiving offers from the energy provider agents, the mediator agent selects which offer(s) is (are) to be selected to create an agreement. First, if in step two a negotiation request was split into multiple requests, a mediator agent combines the offers into one or more possible combinations that satisfy the original negotiation request. If more than one offer remains, the offers are compared, and the best offer is selected. For comparing and selecting offers, the mediator has knowledge enabling comparison of the utility of the energy resources specified in the different offers.

B: Mediator Agent Negotiation Policies

In addition to knowledge concerning energy resources, negotiation policies play an important role in the decision making process of the mediator agent.

B1. Combining Advertisements

Negotiation policies within the mediator agent influence the way in which advertisements are combined. The main purpose of advertisements is to provide consumer agents with information on the available energy resources, and the negotiation constraints over these resources. Policies regulate the amount of combination that occurs within the combined advertisements, by prescribing (i) which energy provider agent advertisements may or may not be combined, and (ii) the number of options that the consumer agent is allowed to see.

B2. Selecting Energy Provider Agents

The energy provider agent selection process is also subjected to policies. These policies direct the selection process, by prescribing constraints on (i) which energy provider agents are allowed to provide combined offers together, and on (ii) the number of energy provider agents that may participate in a negotiation.

B3. Selecting Negotiation Offers

Negotiation offers made by energy provider agents are also subject to selection policies. These policies prescribe which combinations of energy provider agents are (not) allowed to provide combined offers, and which are to be selected in the case of multiple, similar offers.

4.2.3 Provider Agent

Energy resources are encapsulated by energy provider agents. Energy provider agents represent a diversity of energy providers, both small and large, such as consumer homes, universities, or power plants, wishing to sell off excess generated energy. For each resource offered, an energy provider agent implements a function that models the available negotiation space for that resource, based on the constraints of the resource. Such functions may consist of simple price-demand curves, but may also include additional dimensions, such as time-of-day or weather conditions, depending on the type of energy resource being modeled. Provider-level policies also influence negotiations: Policies prescribe additional constraints on advertisement content, and negotiation offers. Based on the resource models and negotiation policies, an energy provider agent constructs advertisements, and handles negotiations for energy resources.

4.2.4 Energy Resource Description

Energy resource descriptions are used to represent the different types of energy resources (such as diesel generators, wind generators, solar arrays, etc.) within the negotiation model. Each resource specifies an amount of energy it can provide. Energy is considered a *divisible* resource: The capacity provided by an energy resource can be divided and allocated to multiple consumer agents. Similarly, energy capacity provided by multiple resources can also be combined.

```
<xs:element name="EnergyResource" type="energy:EnergyResourceType"/>
<xs:complexType name="EnergyResourceType">
 <xs:sequence>
  <xs:element ref="energy:Amount"/>
  <xs:element ref="energy:Price"/>
  <xs:element ref="energy:Reliability"/>
 </xs:sequence>
 <xs:attribute name="resource-id" type="xs:string"/>
 <xs:attribute name="resource-type" type="xs:string"/>
</xs:complexType>

<xs:element name="Amount" type="xs:positiveInteger"/>
<xs:element name="Price" type="xs:positiveInteger"/>
<xs:element name="Price" type="xs:float"/>
```

Example 4.1: Energy resource type definition.

Energy resource descriptions are based on the specification shown in Example 4.1. The elements in the specification define the negotiable issues with respect to an energy resource. The specification allows the expression of requirements/availability concerning *Amount*, *Price*, and *Reliability* of energy capacity. The specification can be extended to include other concepts concerning energy resources. Example 4.2 shows an example of its use. The energy resource definition contains five information elements:

- *resource-type:* The type of resource being negotiated (e.g. diesel-generator, solar array, etc.).

- *resource-id:* A unique identifier for a specific energy resource instance. In most cases, this will not be used in negotiations at the consumer agent level, but can be used by energy provider agents to explicitly map energy resources to specific negotiations.
- *Amount:* Contains the energy capacity that is being negotiated.
- *Price:* Contains the price of the energy capacity that is being negotiated.
- *Reliability:* Contains the reliability of the negotiated energy capacity.

```
<EnergyResource resource-id="diesel1" resource-type="diesel_generator">
  <Amount>350</Amount>
  <Price>12</Price>
  <Reliability>99.85</Reliability>
</EnergyResource>
```

Example 4.2: Energy resource description example.

4.3 Framework Application

In this section, the negotiation framework is applied in two scenarios, each implementing different negotiation types. Scenario A describes how the framework can be instantiated to enable competition between providers, hidden from the consumer agent perspective. Scenario B demonstrates competition between mediator agents, where each mediator agent represents a group of energy provider agents, and consumer agents can choose between multiple negotiation offers. For each scenario, the negotiation policies are described that are implemented in each of the different negotiation participants, to achieve the desired competitive behavior. Scenarios A and B are fully implemented using a prototype of the negotiation framework integrated into the AgentScape system. Simulations are performed and the results are discussed to examine the achieved negotiation behavior.

4.3.1 Scenario A: Provider Competition

In scenario A, the emphasis is on the role of the mediator agent in the negotiation process: The scenario demonstrates (i) how consumer agents defer energy negotiations to a mediator agent, and (ii) how the mediator agent negotiates on behalf of a consumer agent with the available energy provider agents, creating competition between the energy provider agents. The scenario models a world in which energy providers are not directly available for negotiation, but are instead negotiated with by means of a mediator agent, acting as a representative of the parties in the negotiation process. This situation can occur for example when the provider population is too large or too dynamic for a consumer agent to obtain a complete and up-to-date overview of the population. Mediators acts as relatively static entities within this environment, keeping track of the changes in the population. Energy providers and consumers are modeled as highly dynamic entities, with constantly

changing energy supply and demand. The scenario presents example negotiation policies for energy consumer agents, energy provider agents, and mediator agents, which are implemented using the prototype negotiation framework. In this section, the basic scenario design is introduced first. After this, implementation details and examples of actual simulations of this scenario are presented and discussed.

Provider agents are used to represent energy providers that compete for requests for energy. Energy consumers are represented by consumer agents that issue negotiation requests to the mediator agent. The mediator agent negotiates with the individual energy provider agents, and returns negotiation offers to the consumer agents. The energy negotiation model as presented in Section 4.2 is used to model the participants and interactions.

Scenario Outline

Energy consumer agents have varying electricity demand over a certain time period that needs to be satisfied. Energy provider agents are very heterogeneous (i.e., have varying energy capacity), and are willing to deliver energy at a certain price, according to some pricing model internal to each of the agents. In this scenario, a single day is considered. Each hour of this day, negotiations are performed for the allocation of energy capacity in the subsequent hour. Negotiations consist of a single round in each hour. Available energy capacity and energy demand are based on data obtained from a distributed power optimization modeling framework named HOMER[3]. The HOMER framework can be used to model energy technologies, including power sources such as wind turbines, generators, and fuel cells, as well as to model energy loads using daily profiles. For more details on how the framework is used to provide data, see the descriptions of the agents below.

Energy Consumer Agent

Energy consumer agents negotiate for delivery of energy throughout a full day (i.e. 24 hourly negotiations). Energy consumer agents are initialized with a vector specifying the energy demand for the duration of the simulation. This data is obtained from the HOMER framework by specifying an energy demand pattern over the course of a single day. This patterns is based on the typical consumer demand curve as shown earlier in Figure 4.2. The original demand curve is shown in Figure 4.4. Energy demand starts at 7:00, and ends at 23:00, and two large demand peaks are present at the start and end of the day. A smaller demand peak is defined at 13:00. Based upon this default pattern, HOMER generates varying daily energy demand data (one demand value for each hour), for a full year. Each energy consumer agent is given a vector containing the energy demand data from a random day of this pool of 365 days. Figures 4.5, and 4.6 show examples of demand patterns generated from the default demand pattern using the HOMER framework.

Based on its demand pattern, each agent determines whether energy is required in the next hour. If this is the case, a negotiation request is communicated to the mediator agent. The request consists of an energy resource description containing the requested amount of energy, as well as the desired duration and starting time of the delivery of energy. Example 4.3 shows an example of a request. In this scenario, negotiations are focused on

[3] http://www.nrel.gov/homer, National Renewable Energy Laboratory (NREL)

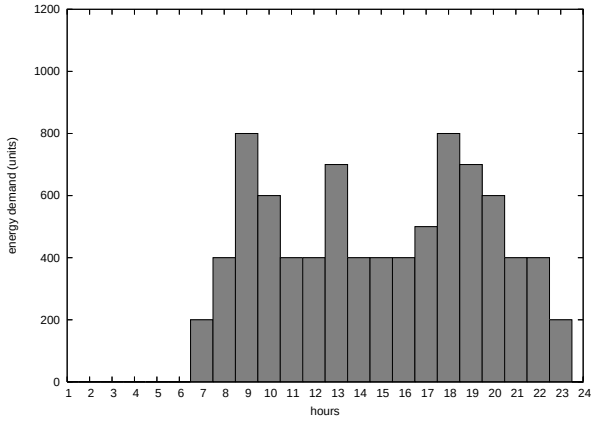

Figure 4.4: The original energy demand pattern from which the consumer agent energy demand is derived.

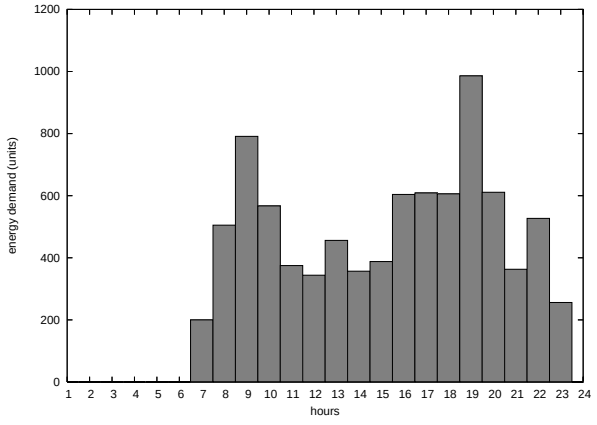

Figure 4.5: Example demand pattern generated using the HOMER framework.

the *price* issue, which represents the *price per unit of energy* (ppu). Reliability of energy is not considered in scenario *A*, see scenario *B* for negotiations that include this concept.

When an offer is subsequently returned by the mediator agent, the consumer agent determines whether it is willing to pay the ppu that is specified in the offer, or whether the proposed ppu is too high. In this scenario, three types of consumer agents are distinguished, each with a different mechanism to determine the maximum acceptable ppu. The three types were chosen to model different types of consumer behavior: Unbounded

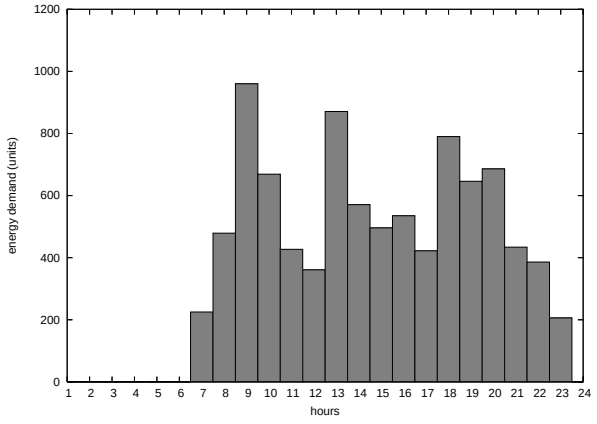

Figure 4.6: Second example demand pattern generated using the HOMER framework.

```
<Agreement>
 <Name>Request_ConsumerAgent1</Name>
 <Context>
  <Duration>1</Duration>
  <Start>13</Start>
  ...
 </Context>
 <Terms>
  ...
    <ServiceDescriptionTerm ServiceName="energy">
     <EnergyResource>
      <Amount>350</Amount>
      <Price></Price>
     </EnergyResource>
    </ServiceDescriptionTerm>
  ...
 </Terms>
</Agreement>
```

Example 4.3: Example of an energy negotiation request in Scenario A.

consumers that always accept the offer made by the mediator agent, strictly bounded consumers that deny any offers above a certain threshold, and adaptive consumers that adapt their threshold during negotiations.

- (A) Fixed maximum ppu: Agents that are equipped with this mechanism determine their maximum acceptable ppu randomly in some predefined range, at the start of the simulation. During the course of the simulation, the agents do not deviate from this value.
- (B) No maximum ppu: Agents equipped with this mechanism have no upper limit, but simply accept the offer that is given to them by the mediator.
- (C) Adaptive maximum ppu: Agents equipped with this mechanism change their

maximum acceptable ppu depending on the success of previous negotiations: Based on the ratio of the amounts of energy that have been successfully obtained and that should have been obtained, the ppu is changed. If the ratio is too low, the maximum acceptable ppu is increased. If the ratio is high enough, the ppu is decreased. As an example, assume the desired ratio is 0.80 (meaning that at least 80% of the demand should be obtained). If the actual ratio is 0.50 (only 50% has been obtained), the maximum acceptable ppu is increased with the difference of these (0.30), resulting in the ppu being increased in this example to 130% of the original value.

Of the three types, the adaptive type can be said to model 'normal' consumer behavior, while the other two agent types model boundaries of consumer behavior. To verify the energy negotiation framework under these different demand models, simulations have been performed using the three agent types. The offer selection policy of each of the abovementioned agents is the same: Any offer that is received that contains a ppu value below the maximum acceptable ppu is accepted by the consumer agents.

Energy Mediator Agent

The mediator agent forwards incoming negotiation requests from energy consumer agents to all of the available provider agents. Based on their responses, the mediator agent in this scenario implements the following offer selection policy:

- If multiple offers are returned which supply the energy originally requested by the consumer agent, select the offer with the lowest energy ppu from these offers.
- If none of the returned offers supply the requested amount of energy, the mediator agent performs a search for a set of agreements that can be combined to deliver the requested amount. This is a combinatorial problem which grows exponentially with the number of available offers. To limit computation time, the first valid combination that satisfies the requested energy amount is returned after a specified time-period, regardless of the ppu specified in these offers.

The selected offer (or offer combination) is subsequently forwarded to the consumer agent. The other offers from the energy provider agents are immediately rejected by the mediator agent. Upon acceptance (or rejection) of an offer by a consumer agent, the mediator agent contacts the involved energy provider agents, and forwards the notification.

Energy Provider Agent

In the scenario, each hour over the course of a day, energy provider agents receive negotiation requests from the mediator agent, to supply energy to consumer agents in the subsequent hour. To simulate dynamic energy providers, each energy provider agent is initialized in a manner similar to the energy consumer agents: A wind turbine model[4] and simulated wind speeds taken from the HOMER framework are used as a basis for energy

[4] the *BWC Excel-R* turbine model is chosen randomly from the available turbine models in the HOMER framework

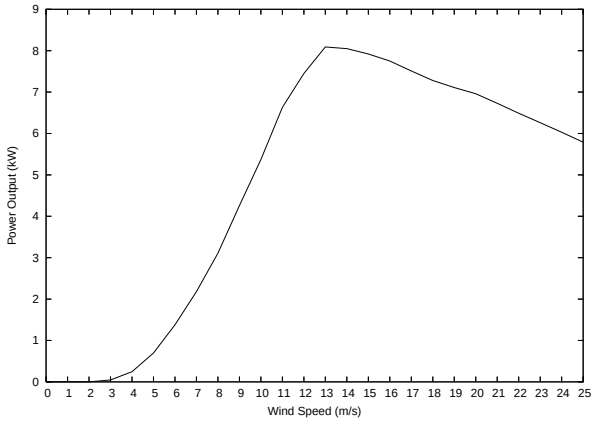

Figure 4.7: The power curve of the wind turbine model used in this scenario.

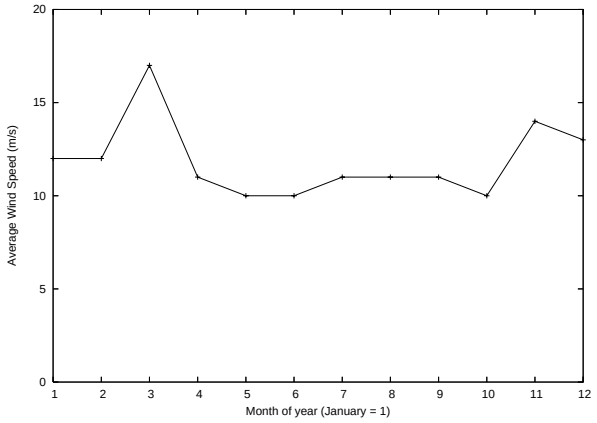

Figure 4.8: The monthly wind speed averages used in this scenario.

supply data. The turbine model provides a realistic power curve indicating the amount of energy generated by the turbine at different wind speeds. Figure 4.7 shows the power curve.

The HOMER framework also provides the functionality to generate wind speed data, using a wind speed distribution model[5]. The wind speeds data generator is supplied with average monthly wind speed values for each month of the year. For this scenario, this

[5]based on a Weibull distribution model

data is obtained from *weatherbase*[6], a source for worldwide monthly weather records and averages. The selected location chosen for this scenario to supply the monthly wind speed averages was Amsterdam, The Netherlands. Figure 4.8 shows the monthly averages used. Based on the wind speed distribution model, and the supplied average wind speeds, 8760 hourly wind speeds values (or: 365 days) are generated by HOMER.

This dataset was used to initialize the individual energy provider agents: Each agent is given a random day from the dataset at the beginning of the day. For this scenario, the data was scaled by a factor of 100, in order to match the amount of energy generated by the individual energy provider agents, to the energy demand of the energy consumer agents. Figures 4.9 and 4.10 show two examples of the power output used in this scenario.

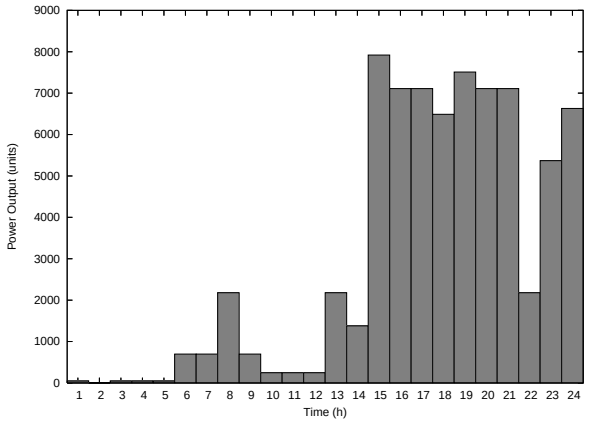

Figure 4.9: Simulated energy supply pattern.

Every hour, energy provider agents determine a new demand price per unit to be used in subsequent negotiations, based on expected available capacity, prior negotiation results, and prior demand price. The overall policy of the provider agents is based around: The main goal of selling at least a minimum percentage of the available energy throughout each day; the concepts of over- and underproduction; whether the energy provider agent is successful in allocating its available energy to consumer agents. The policy specifies that prices are changed only when the energy provider agent is currently in a period of over- or underproduction. If this is the case, prices are increased if the agent has succeeded in selling more than a predefined percentage in the previous round (i.e. energy demand is considered to be high enough). Prices are decreased if this percentage has not been met in the previous round (i.e. energy demand is considered to be low). Also, in the case of decreasing overproduction, prices are lowered in anticipation of less available energy. Furthermore, in the case of overproduction, the base price is lowered by the energy provider agent, to prevent loss of the overproduced energy. The policy is shown in Algorithm 1, in pseudocode notation.

[6]http://www.weatherbase.com

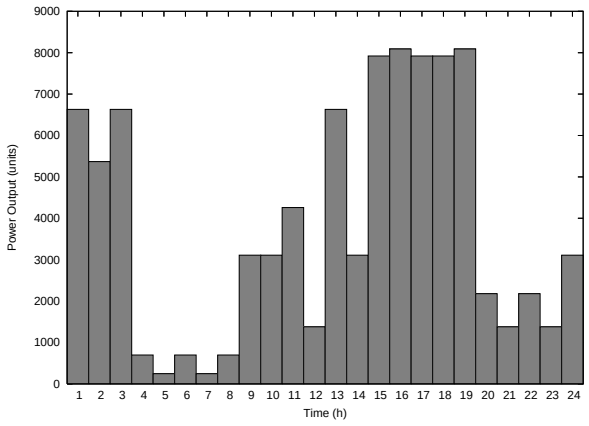

Figure 4.10: Second simulated energy supply pattern.

```
if overproduction then
    new_base_ppu = lower_price_due_to_overproduction(previous_price);
    if previously_also_overproduction then
        if overproduction_is_decreasing then
            new_price = decrease_price(new_base_ppu);
        else
            if previously_enough_sold then
                new_price = increase_price(new_base_ppu);
            else
                new_price = decrease_price(new_base_ppu);
            end
        end
    else
        new_price = new_base_ppu;
    end
else
    new_base_ppu = previous_price;
    if previously_also_no_overproduction then
        if previously_enough_sold then
            increase_price(new_base_ppu);
        else
            decrease_price(new_base_ppu);
        end
    else
        new_price = new_base_ppu;
    end
end
```

Algorithm 1: Pseudocode description of energy provider request handling policy.

4.3.2 Implementation Details and Simulation Results

Scenario A has been implemented using the prototype negotiation infrastructure available within the AgentScape framework: Consumer agents, service provider agents, simulated energy resources, and a mediator agent have been implemented. Furthermore, the domain-specific negotiation policies presented for each of the negotiation participants in the previous section have been implemented. For more details about AgentScape, and the implementation of the negotiation framework in the AgentScape middleware, see Chapter 5. A number of simulations have been performed using this implementation. Examples of simulation runs are presented and discussed in this section, with the aim of demonstrating the energy negotiation framework, and the competition created between the individual energy providers by the mediator agent, under varying consumer demand conditions.

The simulations have been performed using the DAS-2 cluster at the Vrije Universiteit Amsterdam, using the prototype negotiation infrastructure implemented in the AgentScape framework. As this scenario requires the negotiation participants to be aware of the simulated time of day, a virtual clock is implemented to synchronize the time of day between the participating agents.

The following configuration is used in the simulations: 60 energy consumer agents and 15 energy provider agents are instantiated, and a single mediator is instantiated, through which the negotiation parties interact. Energy provider agents are instantiated with a initial price per unit of 30. For each of the three different consumer agent types, the initial maximum acceptable prices are:

- *agent type A (fixed maximum)*: Value is determined randomly in the range 15-30.
- *agent type B (no maximum)*: No maximum value.
- *agent type C (adaptive maximum)*: Initial value is determined randomly in the range 15-30.

Energy supply and demand data are generated prior to the simulations, using the HOMER framework, as described earlier. Results of simulations for each of the different agent types are shown below. The results discussed below are of typical simulation runs, to demonstrate dynamic negotiation behavior of the negotiation model, and do not aim to present the average or overall negotiation behavior of the framework.

Simulation A1: Type A agents

In this simulation, the energy consumer agent population consists of type A agents. In figure 4.11, a typical simulation is shown. Each of the 60 consumer agents has established a maximum acceptable price per unit of energy at the beginning of the simulation. This has resulted in an average maximum acceptable ppu threshold as shown in the figure. Furthermore, the figure shows both the average normalized *offered price* and the average normalized *selling price* of the energy provider agents. The first represents the price for which the energy has been offered during negotiation, the second represents the price for

which energy has been obtained (i.e. agreements have been established) by the consumer agents during negotiation. Both have been normalized with respect to the amount of available energy at that point in time. Finally, the figure also shows the maximum and minimum offered actual prices at each hour by the energy provider agents, using grey bars. This indicates the spread between these maximum and minimum values, which can be interpreted as the bandwidth within which the energy provider agents offer their energy to the consumer agents.

The figure shows that the average normalized selling price is close to the lower end of the offered price bandwidth. This indicates that the market is a so-called *buyer's market* (i.e. buyers determine the price in the market). The increase in the offered and selling prices as indicated in the figure (hours 9–11 and 18–20) also clearly indicate the parts of the day in which energy demand increases, and prices increase accordingly. This increase in energy demand leads to energy becoming scarce, which can also be seen by examining the price bandwidth as shown in the figure. In periods of relatively high demand, the bandwidth of the offered prices decreases, indicating less differentiated prices between the energy providers.

Finally, this simulation shows that the fixed maximum acceptable price set by the consumer agents leads the energy provider agents to offer their energy below this threshold. At two points in the simulation (hours 11 and 18), the energy providers attempt to increase their price above the threshold, but immediately reduce their prices in the subsequent hours, due to the consumer agents not accepting these new prices.

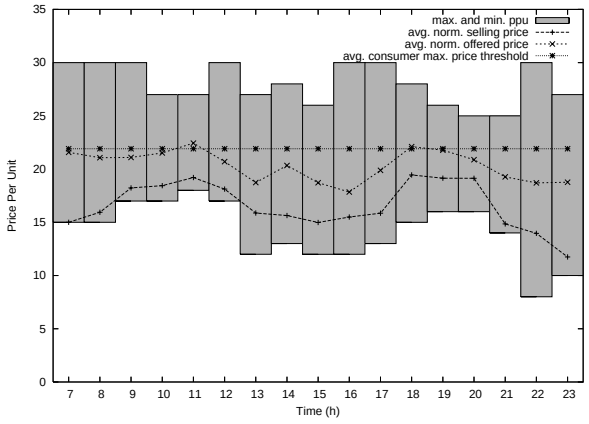

Figure 4.11: Energy negotiation simulation for type A agents.

Simulation A2: Type B agents

In this simulation, 60 consumer agents of type B are negotiating with the available energy provider agents. No fixed maximum acceptable ppu is used by these agents. Figure 4.12 shows the results of a typical simulation run using type B consumer agents. The

figure shows that at hour 9, the average normalized offered price is equal to the average normalized selling price, indicating that all available energy at that time has been allocated. This is due to (i) high energy demand and (ii) the absence of a maximum price threshold in type B agents, which would otherwise have prevented the consumer agents from accepting agreements that are priced too high. Furthermore, the figure shows that in longer periods of relatively high demand (see hours 17-20), the average normalized selling price moves closer to the average normalized offered price, as the consumer agents are forced to accept more expensive agreements, due to high demand. Although consumer agents do not apply a maximum acceptable ppu threshold in this simulation, prices are kept low due to a surplus in energy during most of the day, and sufficient competition between the energy providers. Only during times when energy is relatively scarce, an increase in price occurs.

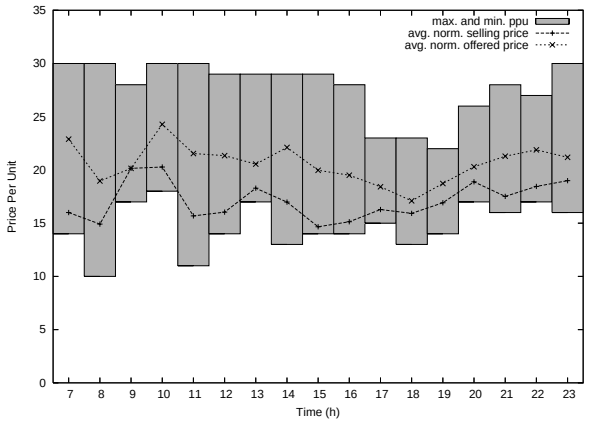

Figure 4.12: Energy negotiation simulation for type B agents.

Simulation A3: Type C agents

In this simulation, the agents adapt their maximum acceptable price, depending on the percentage of required energy that has been successfully obtained. The results of a typical simulation run have been depicted in figure 4.13. In this figure, it can be seen that the consumer agents adapt their acceptance threshold during the simulation. As the threshold is adapted when the ratio between required and obtained energy is above (increase threshold) or below (decrease threshold) .8, the established threshold can be viewed as an indication of a fair price for energy under the current supply–demand conditions in the simulation, if the demand equals to 80% of the available energy.

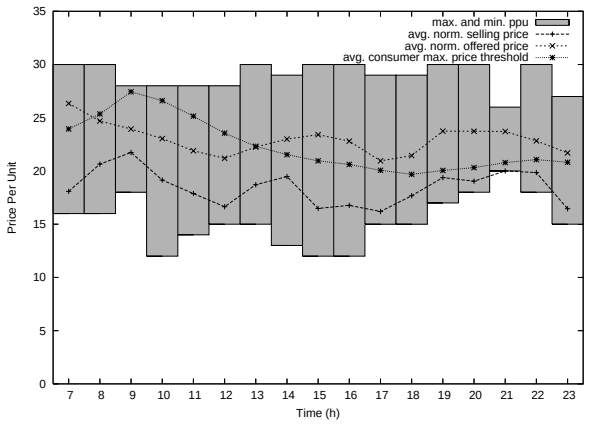

Figure 4.13: Energy negotiation simulation for type C agents.

Scenario Discussion

In this scenario, the basic energy negotiation framework is demonstrated. Results of typical simulation runs are presented, which show the effect of provider competition under different demand conditions (i.e. different agent types, and varying demand over time) on the development of the price in the energy market. The simulations show that under the different supply–demand conditions, the market can be classified as a buyer's market. Even when consumers do not maintain a maximum acceptance price threshold (i.e. type B agents), energy prices are kept relatively low. In part, this is due to the mediator selecting offers on behalf of the consumer agents, which leads to consumer agents only receiving the cheapest offers in the market.

4.3.3 Scenario B: Competition Between Provider Groups

In scenario B, two provider groups exist, one group consisting of very reliable energy providers (e.g. diesel generators), and the other group consisting of less reliable energy providers (e.g. wind generators). The groups can be formed for a number of reasons, for example, groups can represent separate companies consisting of multiple energy providers, or groups can be formed because of a geographical relationship between energy providers. The aim of the scenario is to demonstrate competition between these energy provider groups, by including multiple mediators in the negotiation process.

Scenario Outline

This scenario extends scenario A by including multiple mediators, enabling consumer agents to choose between the negotiation offers made by each of the mediator agents. In addition to the competition within each energy provider group of the individual mediators,

competition is introduced at the level of the mediator agents, as the offers made by each of the mediators are evaluated and compared by the consumer agents.

Additionally, negotiations now include *reliability* of supplied energy, as a second negotiation issue for the negotiation parties to consider during negotiations: Energy consumers do not only have a varying energy demand, but also a varying reliability threshold, indicating the desired reliability of the required energy. Similarly, energy provider agents, in addition to varying power output, simulate varying reliability of the power output. The following sections describe details of the negotiation participants.

Energy Consumer Agent

The energy consumer agents used in this scenario are based on the consumer agents presented in scenario A, with the addition of a component responsible for generating reliability values, and a modified offer selection policy to include the reliability concept in the offer selection process.

In this scenario, energy consumer agents determine the reliability for each negotiation request using a *random walk* mechanism. Each hour during the simulated day, the agents update their desired reliability percentage by increasing or decreasing (direction is chosen randomly) the current reliability value with a predetermined amount. Maximum and minimum value limits are set to constrain the random walk. The desired reliability is included in the negotiation request. Example 4.4 shows an example of a request.

```
<Agreement>
 ...
 <Context>
  <Duration>1</Duration>
  <Start>13</Start>
  ...
 </Context>
 <Terms>
  ...
   <ServiceDescriptionTerm ServiceName="energy">
    <EnergyResource>
     <Amount>350</Amount>
     <Reliability></Reliability>
     <Price>82</Price> <!-- 82 % -->
    </EnergyResource>
   </ServiceDescriptionTerm>
  ...
 </Terms>
</Agreement>
```

Example 4.4: Example of an energy negotiation request in Scenario B.

The established reliability value is used in the offer selection policy to determine whether an offer is acceptable. In scenario A, offer selection consisted only of determining whether the price per unit specified in the (single) offer from the mediator was below the established threshold. In scenario B, the offer selection policy is extended to allow comparison of multiple offers received from multiple mediators: Offers with a reliability below the desired reliability as established by the energy consumer agent, are selected above offers with a matching or higher reliability, if:

- The offered reliability is not below a predetermined percentage of the desired reliability.

- The offered price is at least lowered with a percentage equal or more than the offered reliability is lower compared to the desired reliability. (e.g. if the offered reliability is 75% of the desired reliability, the offered price should be equal to, or less than 75% of the price in the offer to which it is compared (the offer with a reliability equal or higher than the desired amount).

The above selection policy allows an agent to select a cheaper offer if the reliability of the offer is still within acceptable limits.

Energy Mediator Agent

The mediator agents used in scenario B do not differ from the mediator agent in scenario A, with the exception that multiple mediators are now available, each responsible for a separate group of energy provider agents. The offer selection policy employed by the mediators only considers offer price, and does not distinguish on offer reliability. This is left up to the consumer agents.

Energy Provider Agent

Two types of energy provider agents are distinguished in this scenario: Constant and variable. Variable provider agents are similar to the provider agents used in scenario A, but with the extension that provider agents now include a reliability concept for the energy offered to consumers:

- Based on the difference between current and average power output values, a reliability value is calculated at hourly intervals. Two value ranges are distinguished, based on the standard deviation of the power output: When the current output is within the standard deviation range, reliability is lowered linearly from 100% to 80%, based on the distance of the current power output value from the output average. When the current output is between 1 and 2 times the standard deviation range, reliability is lowered linearly from 80% to 40%.

- The calculated reliability value is included in the negotiation offers returned to the energy consumer agents through the mediator. The policy used to determine the offer price is the same policy used in scenario A, and does not include reliability as a factor.

Constant provider agents have a constant power output, and a reliability value of 100%. The policy to determine the offer price is the same as the policy used by the variable provider agents.

4.3.4 Implementation Details and Simulation Results

In this section, two simulation runs are shown and discussed. The simulations use the following configuration: 60 energy consumer agents, 2 mediator agents, and 14 energy provider agents. The energy provider agents are divided equally between the two mediator agents. Furthermore, one group consists of variable energy providers, with varying power output and the aforementioned added varying reliability. The other group consists of energy provider agents with a constant power output. The results of two typical simulation are shown in Figures 4.14 and 4.15.

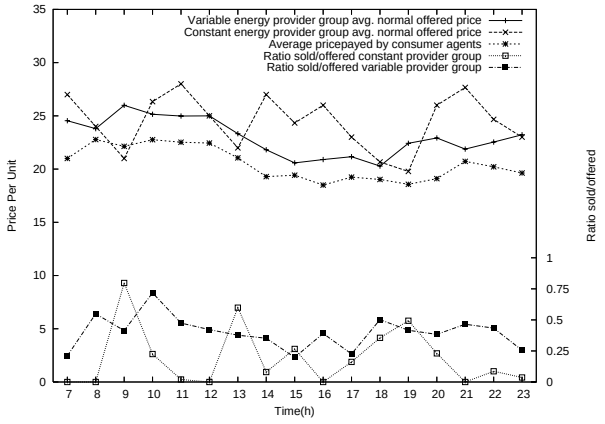

Figure 4.14: Energy negotiation simulation scenario B.

In both figures, the average normalized offered prices of both groups separately, and the average price paid by the consumer agents are shown. The two lower lines in the figures indicate the ratio of the amounts of energy sold versus the amount offered, for the two groups.

In both simulations, the offered prices of the variable reliability energy provider group are on average lower than the prices offered by the constant reliability group, due to the fact that the variable reliability providers compensate the difference in reliability with the constant reliability providers with a lower ppu. Furthermore, it can be seen that the offered prices of the variable reliability energy providers follow the prices payed by the consumer agents more closely than the prices offered by the constant provider group. Also, the fluctuation of the prices offered by the constant provider group are more pronounced. This can be attributed to the facts that (i) the variable provider group is better able to sell energy in all demand situations (i.e. high and low demand), as can be seen in the lines indicating the ratio of the amount sold versus the amount offered, and (ii) the amount of risk-averseness of the consumer agents (i.e. the amount of reliability that is acceptable for the consumer agents).

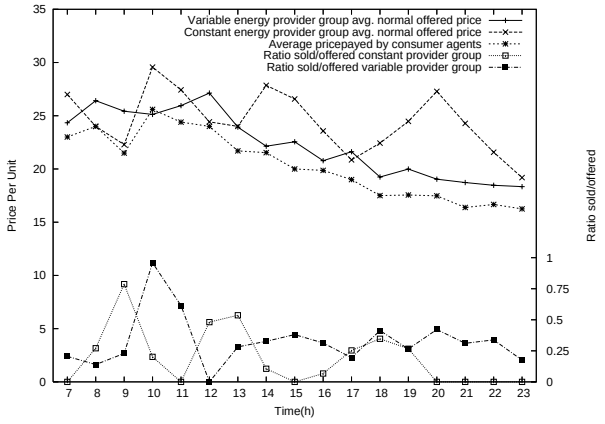

Figure 4.15: Second energy negotiation simulation scenario B.

Scenario Discussion

The negotiation scenario presented in scenario *A* has been extended to include multiple mediators, allowing consumer agents to obtain offers from multiple mediators, creating competition between the provider groups represented by each of the mediators. Furthermore, energy *reliability* has been included in the model as additional negotiation issue. Simulations have been presented that demonstrate mediator competition by differentiating provider groups with respect to the reliability of the energy they are offering. Provider groups offering less reliable energy compensate for this by reducing their demand price, increasing their chance of acceptance. This results in the less reliable provider groups being able to sell available energy more successfully in the varying demand conditions throughout the simulated day.

4.4 Framework Extensions

In this section, extensions to the negotiation framework are presented. Although the framework provides for much flexibility through adjusting the negotiation policies and the negotiation language of the negotiation participants (as demonstrated in the previous section, and Chapter 5), not all negotiation models are captured by our negotiation framework. However, the basic building blocks provided by the framework can be extended to allow for other types of negotiation:

- The negotiation language can be extended to include additional negotiation related information. Note that these extensions are different from the domain-specific negotiation language extensions that are part of the normal framework instantiation process within a particular domain. The extensions considered here are not part of these domain-specific extensions, but are modifications to the basic negotiation

language structure: Additional language elements are defined that can be used by the negotiating parties to express additional negotiation related information during the negotiation process.

- The four basic negotiation phases used at both levels in the negotiation framework can be re-used in different ways, to achieve different negotiation interactions. Negotiation phases on the different levels can be decoupled, to allow for different interactions on each of the levels. Furthermore, phases can be repeated and the role of the phases in the overall negotiation process can be redefined.
- The role of the mediator in the interaction process can be redefined, allowing the mediator to govern the interaction taking place at the two levels in the negotiation process, in order to achieve the desired negotiation interaction behavior.

To demonstrate the application of the framework to other negotiation models, two scenarios are presented in this section. Scenario C demonstrates the extension required to enable competition between energy consumers. For this purpose, the framework is extended to accommodate an auctioning model. Scenario D describes an extension of the framework which allows negotiation participants to specify decommitment penalties. For each of the scenarios, the required extensions to the framework are discussed, and example traces are given to demonstrate the use of the extensions.

4.4.1 Scenario C: Consumer Competition

In scenario C, competition between energy consumers is achieved by implementing an auctioning model using the negotiation framework. Auctions are frequently used as a market form in newly deregulated traditional energy markets. The effects of different auction models on the behavior of electricity markets is the subject of ongoing research [39, 80, 92]. Our auction interaction model is based on the negotiation language and protocol as described in Chapter 3, and supports most common auction models, such as *ascending bid, descending bid, and first- and second-price sealed bid auctions*, as well as bid withdrawal. The model allows for multiple buyers and sellers (double auctions), as well as multiple issue auctions, including package bidding. The role of the mediator in the negotiation process changes from reactive to pro-active, to the role of an auctioneer.

The auction interaction model is described below. After that an example auction is presented.

Auction Interaction Model

Figure 4.16 shows a schematic overview of the steps in the auction interaction model, in which the negotiation framework interactions that take place at each of the two negotiation levels, at different points in the auction process, are distinguished.

An auction starts at the mediator-provider agent level, where the provider agents have specified the auction issues that they wish to sell through the auction. The mediator agent collects this information from the various provider agents. The mediator agent decides in which configuration the issues specified by the provider agents are auctioned (e.g. each issue separately, all issues combined in a single auction, etc.). When this information has

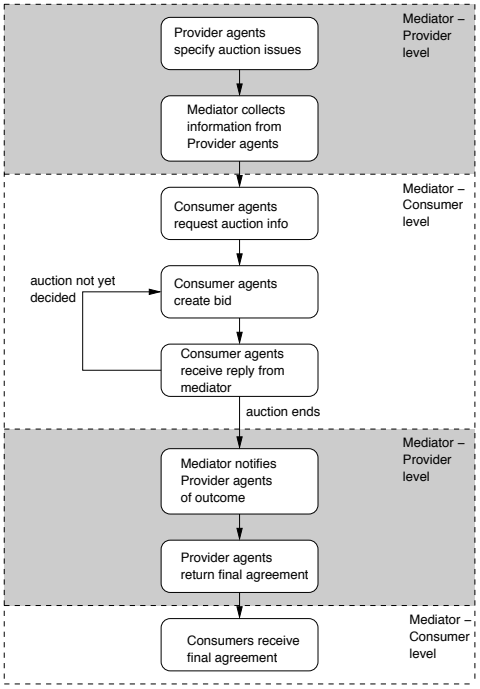

Figure 4.16: The auction interaction model.

been collected, the mediator agent selects the (next) provider for which it will perform an auction. The auction now continues on the mediator-consumer agent level. Upon the start of an auction, consumer agents request the information concerning the auction (which issues are being auctioned, etc.), collected by the mediator agent[7]. After this, a number of auction rounds are performed, during which the consumers agents submit bids to the mediator agent. After the mediator agent has received all bids (or a time-out occurs), the mediator decides whether an additional round is required, or whether conditions are met that signal the end of the auction (e.g. no new bids in the previous n rounds, time limit reached, etc.). When the mediator agent decides that an additional round is required, the consumer agents are informed. If the auction has ended, the mediator agent determines the allocation of the issue(s) to the consumer agents, but does not yet inform the consumer agents of the outcome: The mediator agent first notifies the provider agents of the outcome of the auction. The provider agents then confirm this result. The resulting confirmation is returned to the consumer agents, after which the auction ends.

Although the semantics have changed, the interaction model uses the basic elements of the negotiation language (templates and agreement-request and agreement-offer documents), and implements the four phases of the original interaction model (advertisement, request, offer, acceptance). Contrary to the original interaction model, the negotiation phases at both interaction levels are no longer linked to each other. Figure 4.17 shows the phase transitions on the two levels, and their relationships. Below, for each of the negoti-

[7] we assume that the starting time of auctions is known to the consumer agents

ation levels, the different phases and their place in the auction process are described.

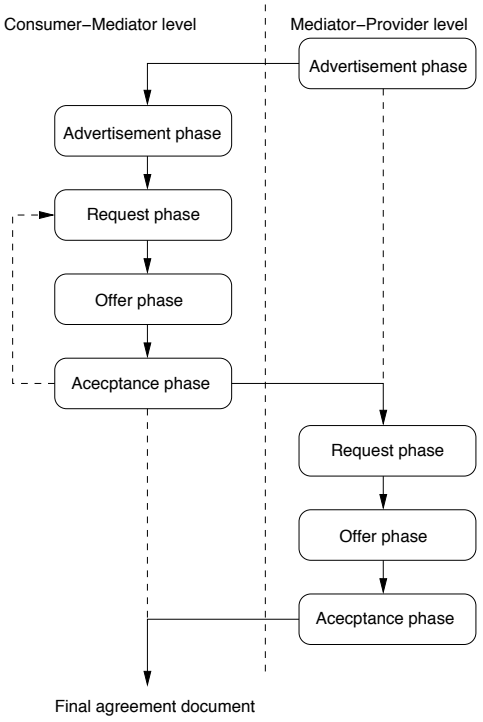

Figure 4.17: The auction phase transitions.

Advertisement Phase

- Mediator-Provider agent level: In this phase, provider agents indicate the items that are made available in the upcoming auction (e.g. the energy capacity that they have available), by specifying this in their template document. The creation constraints section of the document can be used to specify any constraints that a provider agent wishes to apply to the auction. For example, a minimum price can be specified, by defining a minimum value constraint for the price element of the offered service description term. At the start of a new auction, the mediator agent collects these templates from the provider agents.

- Consumer-Mediator agent level: Consumer agents that want to participate in the auction contact the mediator agent, and request the available templates. Based on the auction model implemented by the mediator agent (e.g. simultaneous or sequential), the mediator either bundles all available provider agent templates, or selects one of the available templates, and returns it to the consumer agents. The template contains all available auction item information, including available item packages if the auction supports these. Term compositors are used to indicate item packages.

Example 4.5 shows a template in which two items can only be requested as a combination, indicated by the *All* compositor element. The consumer agents use the template to determine which bidding options are available to them, and adjust their bidding strategies.

```
<Template>
 <Name>AuctionItems_Mediator</Name>
 <Context>...</Context>
 <Terms>
  <All>
   <ServiceDescriptionTerm serviceName="serviceA">...</ServiceDescriptionTerm>
   <ServiceDescriptionTerm serviceName="serviceB">...</ServiceDescriptionTerm>
   <All>
 </Terms>
</Template>
```

Example 4.5: Example template document containing packaged auction items.

Request Phase

- Mediator-Provider agent level: The mediator agent informs the energy provider agents which consumer agents have been awarded the auction items. A negotiation request document is communicated to the energy provider agents, describing which auction items are awarded, including the final price established during the auction.
- Consumer-Mediator agent level: Consumer agents communicate bids to the mediator agent, for the auction items specified in the template information. For each item on which a consumer agent wishes to bid, a bid is specified in the agreement request document. If a consumer agent decides that its standing bid is still satisfactory, the standing bid is sent to the mediator agent, to indicate no change is required. Consumer agents can also withdraw standing bids in this phase, by removing the standing bid for that item.

Offer Phase

- Mediator-Provider agent level: Provider agents acknowledge the notification received from the mediator agent, by returning an agreement offer document detailing the agreement that is to be established as a result of the auction.
- Consumer-Mediator agent level: The mediator agent informs the consumer agents about the progress of the auction. The mediator agent determines the current state of the auction based on the pricing rules it implements (ascending bid, descending bid, etc.), and chooses the amount of information it reveals in the offer, to allow for example anonymous bid information to be distributed.

Acceptance Phase

- Mediator-Provider agent level: If an auction has been successfully closed, the mediator agent establishes the final agreements by accepting the agreement offers received from the provider agents. The provider agents implement the accepted offer, and return a final agreement document describing the allocation of the awarded items to the specific consumer agents.
- Consumer-Mediator agent level: The consumer agents indicates whether the auction information as was received in the previous offer phase, is considered to be correct. If an agent decides to continue participating in the auction, the mediator agent is informed of acceptance. However, a consumer agent can also decide in this phase to withdraw from the auction entirely. As a result, the mediator agent removes the consumer agent's current bid, and removes the agent from the auction in the next round. The mediator agents examines the current state of the auction, and decides whether an additional round is required. If this is not the case, the auction is ended by contacting the provider agents to obtain the final agreements, and creating final agreement documents for each of the participating consumer agents. If an additional round is required, an agreement document is returned containing the current state of the auction, and an indication the auction has not yet resulted in a final agreement, and that an additional round is required.

The next section presents an example auction scenario which shows the interactions required to implement an auction framework, using our negotiation framework. After this, an example auction is presented to demonstrate the use of the framework.

4.4.2 Auction Framework Example Trace

To demonstrate the adaptation of the framework to the auction domain, this section presents an example auction sequence. The auction type chosen in this example is a simultaneous ascending auction, as this auction type has relatively straightforward rules (bidders bid for items they are interested in, see all highest bid prices, and pay their bid if awarded), and is considered a suitable auction type for auctions concerning substitute goods [28, 66]. The basic simultaneous ascending auction model does not support package bidding (for an approach which extends the model to include package bidding. see [29]). In this scenario, any bidders interested only in packages of items can use bid withdrawal to withdraw any bids if it becomes clear that not all items are attainable. An example of the bid withdrawal mechanism as provided by our auction framework is included in the trace. Energy resources that are offered are substitute goods (i.e. in our scenario, items only differ in energy amount offered, no additional issues such as reliability are considered). The market is a per-day market, i.e. each auction concerns the energy that will be delivered the following day. The example auction consists of two energy provider agents offering energy, and three energy consumer agents acting as bidders. A mediator agent implements the role of auctioneer. Below, a number of rounds of the auction will be discussed, to demonstrate the framework interactions.

```
<Template>
 <Name>AuctionItems_ProviderA</Name>
 <Context>...</Context>
 <Terms>
  <ExactlyOne>
   <All>
    <ServiceDescriptionTerm serviceName="energy">
     <EnergyResource resource-id="dieselA" resource-type="diesel˙generator">
      <Amount>450</Amount>
      <Price></Price>
     </EnergyResource>
    </ServiceDescriptionTerm>
   </All>
  </ExactlyOne>
 </Terms>
 <CreationConstraints>
  <Item>
   <Location>
    //Context/Price
   </Location>
   <Restriction type="minValue">
    <minValue>2000</minValue>
   </Restriction>
  </Item>
 </CreationConstraints>
</Template>
```

Example 4.6: Template document created by Provider A.

Round One

Before the start of the first round, the energy provider agents create their templates, describing the offered energy. In this example, provider agent A offers 450 units of power, and defines a constraint regarding this offer that indicates the minimum price the provider agent wishes to receive. Example 4.6 shows the template specified by provider A. Provider agent B specifies a similar template, offering 250 energy units, and a minimum price of 1100.

At the start of the first round, the mediator agent collects the templates. The mediator agent bundles this information into a single template document. The mediator agent uses the template term composition structure to indicate which bid possibilities are available. In our scenario, consumer agents can either bid on one or both of the items separately. The creation constraints used by the provider agents to indicate reserve price are not included in the template which is sent to the bidding agents, as this information is used by the mediator agent in the auction process, and is not intended to be known to the bidding agents. Example 4.7 shows the structure of the template document created by the mediator agent.

```
<Template>
 <Name>AuctionItems_Mediator</Name>
 <Context>...</Context>
 <Terms>
  <OneOrMore>
   <ServiceDescriptionTerm serviceName="energy">
    <EnergyResource resource-id="dieselA" resource-type="diesel˙generator">
     <Amount>450</Amount>
     <Price></Price>
    </EnergyResource>
   </ServiceDescriptionTerm>
   <ServiceDescriptionTerm serviceName="energy">
    <EnergyResource resource-id="dieselB" resource-type="diesel˙generator">
     <Amount>250</Amount>
     <Price></Price>
    </EnergyResource>
   </ServiceDescriptionTerm>
  <OneOrMore>
 </Terms>
</Template>
```

Example 4.7: Template document created by mediator agent, containing two auction items.

Every consumer agent requests the available template information from the mediator agent, and receives the bundled template as a result. Based on this template, the consumer agents make their first bids, by constructing a negotiation request document containing the bids for the items they are interested in. The request contains one of the options presented in the template, for which the consumer agent has filled in the 'Price' element of the document. In this scenario, consumer agent 1 bids on both items, and intends to obtain both, consumer agent 2 bids only on the item of provider agent A, and consumer agent 3 bids on both items separately, in an attempt to acquire either one, or both of the items. Example 4.8 shows the request created by consumer agent 1.

```
<Agreement>
 <Name>Request_Consumer1</Name>
 <Context>
  <AgreementInitiator>Consumer_1</AgreementInitiator>
  <TemplateName>AuctionItems_Mediator</TemplateName>
 </Context>
 <Terms>
   <All>
    <ServiceDescriptionTerm serviceName="energy">
     <EnergyResource resource-id="dieselA" resource-type="diesel˙generator">
      <Amount>450</Amount>
      <Price>2100</Price>
     </EnergyResource>
    </ServiceDescriptionTerm>
    <ServiceDescriptionTerm serviceName="energy">
     <EnergyResource resource-id="dieselB" resource-type="diesel˙generator">
      <Amount>250</Amount>
      <Price>900</Price>
     </EnergyResource>
    </ServiceDescriptionTerm>
   <All>
 </Terms>
</Agreement>
```

Example 4.8: Example agreement request in round 1.

The mediator agent analyzes the incoming bids, and determines whether an additional round is available (i.e. if new bids have been received). Figure 4.18 shows these interactions taking place in this first round of this scenario.

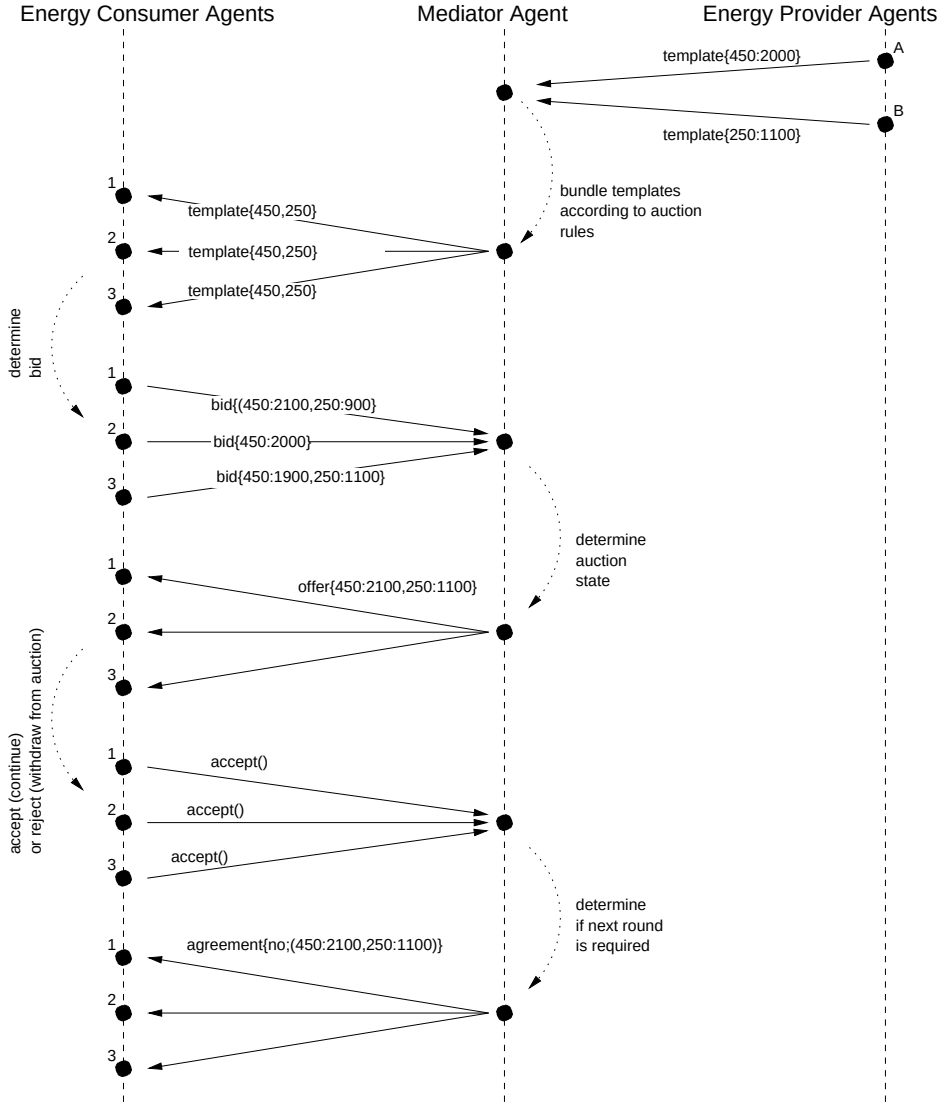

Figure 4.18: The first auction round.

Round Two

In round two, consumer agent 1 decides to increase its bid on the 250 item to 1200, in an attempt to obtain the highest bid on both the auction items. Consumer agent two however,

also increases its bid on the 450 item to 2200. Consumer agent 3 however, based on the bids in the previous round, decides to forego bidding on the 450 item, and withdraws its bid on the item from the auction, by removing the bid for this item from the negotiation document it will send to the mediator agent. The bid on the 250 item remains unmodified. Finally, all the agents send their new bids to the mediator agent.

As a result of the new bids, the mediator agent determines the new state of the auction, and returns this to the consumer agents. The agents accept the current state of the auction. Figure 4.19 shows the interactions taking place in round two.

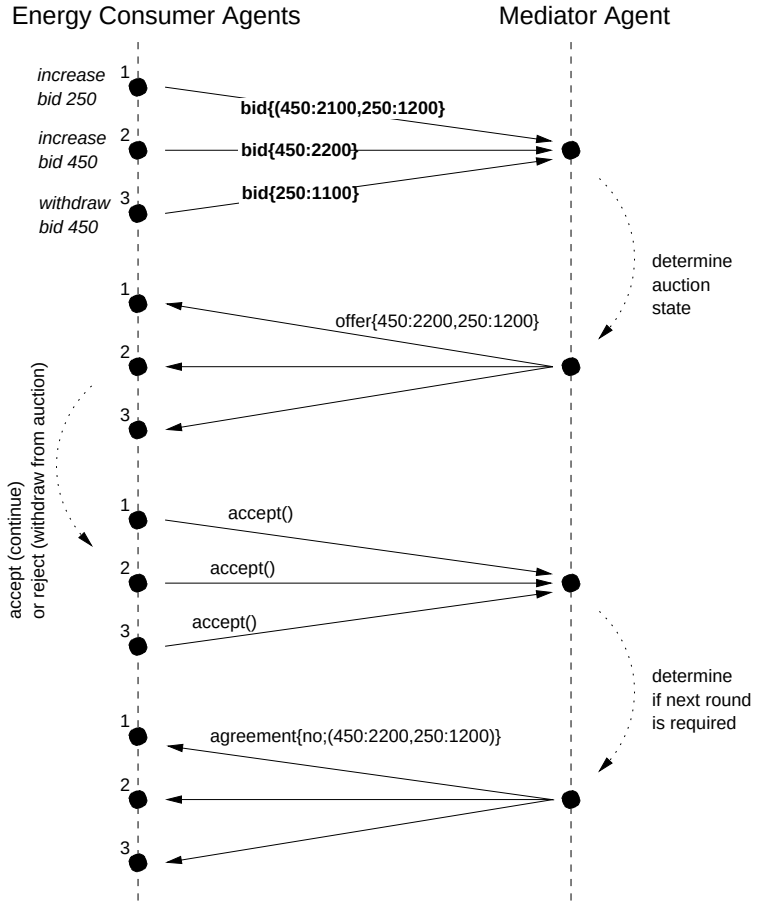

Figure 4.19: The second auction round.

Round Three

In round three, consumer agent 1 decides to take no action, and await the bids of the other agents in this round. Consumer agent 2 increases its bid on the 450 item to 2300, thereby outbidding consumer agent 1. Consumer agent 3 also decides to await the bids of the other agents in this round. All agents send their bid information to the mediator agent.

The mediator agent determines the new high bids for the items, and communicates this to the consumer agents using the agreement offer document. As a result, consumer agent 1 decides that it will not succeed in obtaining the 450 item, as the bid made by consumer agent 2 is too high. As consumer agent 1 was only interested in both items together, it decides to withdraw its bids on all items, and leave the auction. The agent indicates this to the mediator agent by issuing a reject back to the mediator agent. The mediator agent withdraws the standing high bids the consumer currently has in the auction[8]. The current state of the auction is returned to the consumer agents, indicating the new standing high bid on the 250 item, due to the withdrawal of consumer agent 1. Figure 4.20 shows the interactions taking place in round three.

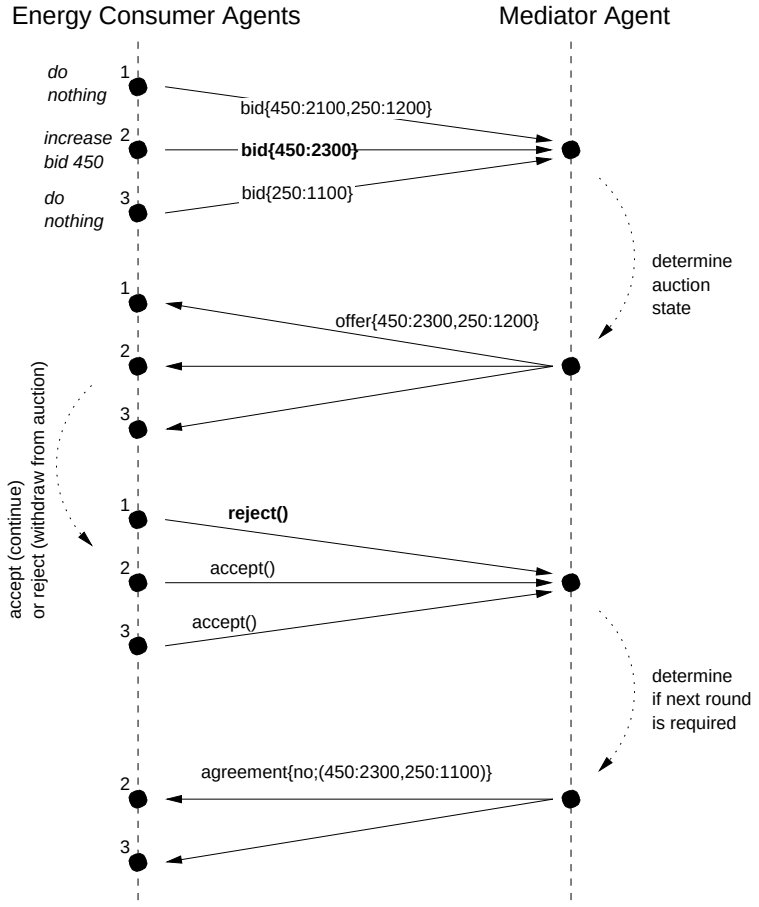

Figure 4.20: The third auction round.

Rounds continue in the manner described above, until the final round is reached.

[8] A standard penalty mechanism for withdrawal may be used by the mediator agent as a result of the withdrawal: The withdrawing bidder is responsible to pay the maximum of 0 and the difference between the withdrawn bid and the final sale price.

Final Round

In the final round, the mediator agent decides that the current state of the auction is the final state (e.g. due to time constraints, or due to a number of previous rounds in which no new bids were received), and contacts the energy provider agents to create the agreement documents for the consumer agents. The energy provider agents are sent the final state of the auction, for each of the items they have brought in the auction. As a result, the energy provider agents inform the mediator agent about the agreement that will be created if accepted by the mediator (i.e. acknowledgment of the received request). The mediator accepts the offers from the energy provider agents, and receives the final agreement documents in return. Finally, the agreement documents are sent to the energy consumer agents in response to their acceptance of the auction state. Figure 4.21 shows the interactions taking place in the last round[9].

[9]The withdrawal penalty which is issued to consumer agent 1 due to the withdrawal of its bid on the 250 item in round three is not shown here.

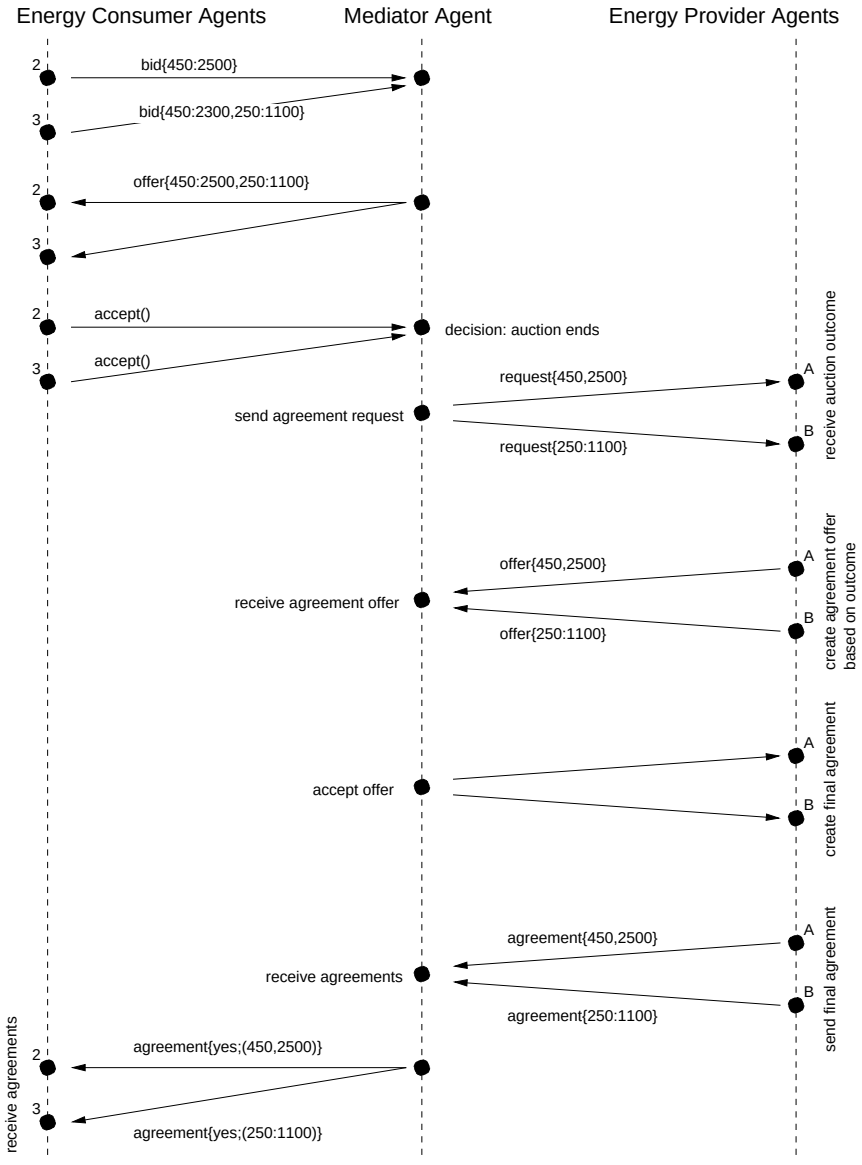

Figure 4.21: The final auction round.

4.4.3 Scenario Discussion

In this scenario, our negotiation framework was adapted to support competition between consumers in the negotiation process. An auction-like approach was chosen, in which the mediator was assigned the role of auctioneer. A *Price* element was explicitly incorporated in the negotiation documents, to enable consumer agents to bid for the services offered by the service provider agents. Furthermore, the original framework protocol was adapted to implement auction interactions, by assigning different semantics to the interaction available in the protocol. The mediator is responsible for performing the auction process on behalf of the provider agents. The provider agents provide templates to indicate the items that will be included in the auction, and are contacted again by the mediator agent once the auction has ended, and the auction items have been awarded to the consumer agents. The mediator establishes the final agreements with the provider agents based on the outcome of the auction, and provides the agreement documents to the consumer agents, as final proof of the outcome of the auction. An example auction is presented in which a simultaneous ascending auction was implemented using the auction framework, including bid withdrawal. The auction framework supports several common auction models, as described earlier.

4.4.4 Scenario D: Decommitment

In this section, our negotiation framework is extended to allow participants of the negotiation process to unilaterally end ongoing negotiations. First, a short overview is presented of the various models described in literature. After this, the extensions of the negotiation framework are presented. Finally, an example scenario is presented, demonstrating the extension of the model.

In literature, various methods have been proposed to allow negotiation participants to end ongoing negotiations. Kraus et al. [55] include the notion of time as a driving force in the negotiations. In their model, time is considered valuable. Agents prefer any agreement in any given time period over the indefinite continuation of the negotiation process. Furthermore, agents prefer establishing agreements sooner rather than later. This approach allows agents to end negotiations if a solution has not been found within a certain amount of time. In this approach, agents still commit fully to a negotiation, but negotiation times are shortened, allowing agents to more quickly resolve ongoing negotiations, and engage in other possible negotiations that may present themselves.

Other approaches involve allowing agents to ‘opt out’ of ongoing negotiations. This enables agents not only to reduce negotiation time spent on negotiations that are no longer useful, but also to stop ongoing negotiations if other, more interesting negotiation options have become available. An early example in this area is that of *contingency contracts* (Raiffa 1982). In this approach, contracts are made contingent on probabilistically known future events. If these events arise, an agent may drop its commitments regarding the contract. In realistic settings however, contingency contracts have a number of problems: Not all future events may be known; The number of (relevant) events may be very large preventing an agent from monitoring all events; events may not be observable by all parties, which allows for agents to lie about the events that it has (or has not) monitored.

An approach that is put forward by Sandholm and Lesser is the so-called *levelled commitment contract*. The main idea of this approach is to allow negotiation participants to unilaterally decommit from a negotiation, in exchange for a decommitment penalty. The penalties can be used to choose a level of commitment, ranging from traditional full-commitment contracts, to commitment-free contracts. Advantages of this approach as described by Sandholm et al. [85] include: Accommodation of new domain and negotiation events in the negotiation process; enabling agents to use decommitment to perform a backtracking search in the space of available negotiation options; enabling profitable construction of combinatorial contracts from basic contracts (agents can try to obtain a number of contracts, but drop established commitments when not all contracts can be obtained); reduction of time and computation required for negotiations; up-front feasibility checks are not necessary, as negotiations can be aborted later in the negotiation process, if required. Sandholm et al. show that, using decommitment, agents can reach higher utility levels in case of negotiations with uncertainty about future events.

In addition to the issues under negotiation (usually goods and price), a levelled commitment contract contains two additional elements: The contractor's decommitment penalty, and the contractee's decommitment penalty. When an agent receives a so-called outside offer which is preferred over the current offer, an agent can decommit upon which it receives the penalty specified in the contract. Each agent's decommitment strategy is based on the *decommitment threshold* of the agent.

With respect to decommitment mechanisms, three different leveled commitment mechanisms are distinguished by Sandholm et al.: Sequential decommitment; simultaneous decommitment where both parties pay their respective penalties when both decommit; and simultaneous decommitment where neither party pays a penalty when both decommit. Which mechanism to use depends on the conditions in which the negotiations are taking place, the types of agents under consideration (risk-neutral or not) and the utility functions of the agents. With respect to the number of interactions required, simultaneous decommitment where neither party pays leads to the least amount of interactions, which could be of importance in domains where interactions are difficult or expensive. With respect to complexity: sequential decommitment is easy for the 'second mover': 'decommit' nonstrategically if the first mover did not decommit, and never decommit if the first mover already decommitted first.

Examples for implementing leveled commitment contracts are given by Sandholm et al. in [86], and by Excelente-Toledo et al. in [37]: Both works present algorithms for deciding when to drop a commitment, and algorithms for determining decommitment penalties. In the former work, two algorithms are presented for: (i) Computing the Nash equilibrium[10] decommitment thresholds and decommitment probabilities for rational agents; (ii) optimizing the contract price and penalties to maximize expected payoff. In the latter work, three levels of commitment are described: *Total*, in which agents cannot drop commitments; *loose*, in which agents always drop their commitments once a better offer is received; *partial*, in which agents drop commitments with a percentage of probability if a better offer is received. Furthermore, three mechanisms for determining decommitment penalties are discussed: *Fixed*, in which penalties are defined to be a pre-specified percentage of the reward specified in the contract; *partially sanctioned*, in which the penalty

[10] no agent can improve its result by unilaterally changing its decommitment strategy

is determined dynamically, based on the likelihood of decommitment (high if chance of decommitment is high); *sunk cost*, in which penalties are also dynamic, and increase over time to reflect the effort spent on a contract. These dynamic penalty mechanisms provide a more realistic model for determining the costs related to dropping commitments in an ongoing negotiation process.

Scenario Outline

The adaptation of our framework to include decommitment is presented in the context of the energy management domain, as described earlier in this Chapter. Providing the option to decommit in negotiations in this domain allows for more flexibility in the negotiation process. In the energy management domain, flexibility is important, as changes in the supply or consumption of energy may occur without prior notification, and due to real time constraints, the balance between supply and demand needs to be re-established rapidly, as dealing with energy shortages or surpluses is costly.

Decommitment Interaction Model

The decommitment model implemented in this scenario is a leveled commitment protocol, with penalties for each of the negotiating parties. Penalties are fixed during negotiations, but can be modified between different negotiations, by each of the negotiating parties. This mechanism is comparable to the *partially sanctioned* mechanism mentioned earlier. We consider only decommitment during the negotiation process. After an agreement is established, other mechanisms may be used to determine what happens when an agreement that is active, is violated. This may for example be specified in other agreements, established earlier, which cover agreement violation terms.

To allow for specifying levels of commitment in our framework, the information exchanged between the negotiating parties (i.e. service consumer agents and service provider agents) is extended to include information on decommitment penalties. Both consumers and providers can specify decommitment penalties, indicating to the other party the price they will have to pay if the party decides to back out of the negotiation. Furthermore, the protocol is extended to include an intermediate phase, in which the negotiating parties agree on the negotiation terms (i.e. the penalties specified by both parties), before proceeding with the negotiation.

To allow for the expression and use of decommitment contracts in our framework, a number of extensions to the framework are made, both in the negotiation language, and the negotiation protocol. These changes are discussed below.

Negotiation Language The *Context* section of the negotiation documents (templates and agreement requests/offers) is extended with a *Penalties* section enabling the specification of penalties by the negotiating parties. In the template document, service provider agents can add penalty information to the service options (i.e. the service description terms) described in the template. This allows a service provider agent to specify different penalties for the services or service combinations that are specified in the template. Any negotiations that are initiated by service consumers using this template, can only be

aborted by paying the specified penalty. Each penalty specification refers to a subsection of the service description terms section using an XPath expression. To uniquely identify subsections of the SDT section, an additional attribute can be specified within each term compositor element. Linking penalties to subsections also allows the mediator agent to combine multiple templates into a single template, while retaining the penalty information specific to each provider template on which the combined template was based. Example 4.9 shows an example of a template including penalty information for the different service options. In this example, two options are specified in the template: *provider1_option1* for negotiating a service package consisting of three services, and *provider1_option2*, negotiating for single service. For both options, decommitment penalties are specified. For option 1, the penalty is set higher than option 2, as the risks for negotiating option 1 are higher from the perspective of the provider agent.

```
<Template>
 <Name>Template_Provider1</Name>
 <Context>
  ...
  <Penalties>
   <ProviderPenalty location="//All[@id='provider1_option1']">40</ProviderPenalty>
   <ProviderPenalty location="//All[@id='provider1_option2']">5</ProviderPenalty>
  </Penalties>
 </Context>
 <Terms>
  <ExactlyOne>
   <All id="provider1_option1">
    <ServiceDescriptionTerm serviceName="serviceA">...</ServiceDescriptionTerm>
    <ServiceDescriptionTerm serviceName="serviceB">...</ServiceDescriptionTerm>
    <ServiceDescriptionTerm serviceName="serviceC">...</ServiceDescriptionTerm>
   </All>
   <All id="provider1_option2"">
    <ServiceDescriptionTerm serviceName="serviceA">...</ServiceDescriptionTerm>
   </All>
  </ExactlyOne>
 </Terms>
 <CreationConstraints>
 ...
 </CreationConstraints>
</Template>
```

Example 4.9: Example template containing the penalties extensions.

In an agreement request document, a service consumer agent can specify its decommitment penalty by specifying it in the *Penalties* section of the negotiation request. The agreement request only contains one of the options that were specified in the templates (the option the consumer agents has selected to negotiate for), as the penalty does not refer to a specific subsection of the SDT section, but holds for the entire agreement request. Example 4.10 shows an example agreement request document containing the penalty specified by the consumer agent, in addition to the penalty specified by the provider agent for the service description terms.

Negotiation Protocol

The negotiation protocol follows the same phases as defined in Chapter 3, with the addition of an intermediate acceptance phase, in which negotiating parties agree on the penalties as specified. This additional phase is added after the offer phase, and before the

```
<Agreement>
 <Name>Request_Consumer1</Name>
 <Context>
   <ConsumerPenalty>20</ConsumerPenalty>
   <ProviderPenalty location="//All[@id='provider1_option2']">5</ProviderPenalty>
  <AgreementInitiator>Consumer_1</AgreementInitiator>
  <TemplateName>Template_Provider1</TemplateName>
 </Context>
 <Terms>
  <All id="provider1_option2"">
   <ServiceDescriptionTerm serviceName="serviceA">...</ServiceDescriptionTerm>
  </All>
 </Terms>
</Agreement>
```

Example 4.10: Example agreement request containing the penalties extension.

final acceptance phase. Figure 4.22 shows the phase transitions including the intermediate acceptance phase.

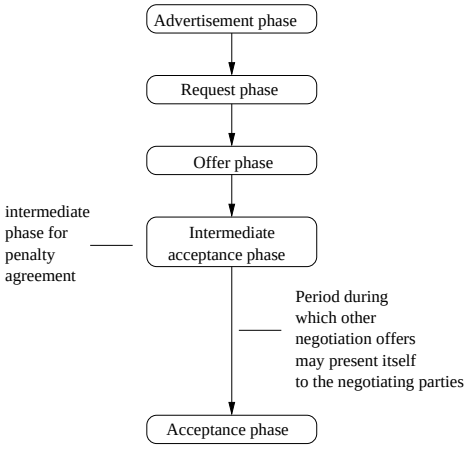

Figure 4.22: The phase transitions in the negotiation model including decommitment.

Consumer Agent In the intermediate acceptance phase, a consumer agent decides (based on the decommitment threshold it has set for itself) whether an offer it has received is accepted, after which penalties have to be paid if the agent decides to reject the offer at a later point in time. If an offer is rejected in the intermediate phase, the negotiation ends immediately and no penalty has to be paid. Figure 4.23 show the possible scenarios for the consumer agent in the extended protocol, and the resulting penalties that are received or paid.

Provider Agent In the extended protocol a provider agent can decide, after receiving a negotiation request, to either immediately end negotiation by returning an empty offer document, or to commit to the received request, by sending a negotiation offer. If the offer is accepted by the consumer agent, an intermediate accept notification is received. Otherwise, a intermediate reject notification is received, in which case the negotiation

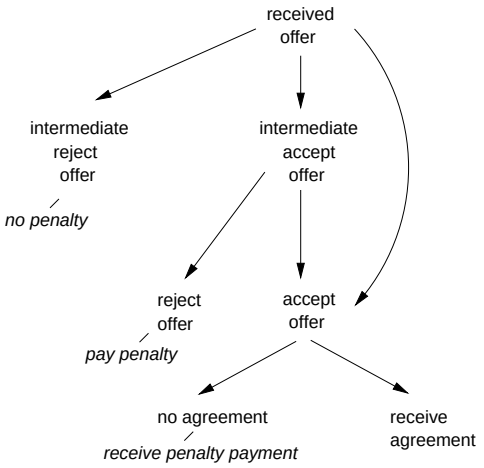

Figure 4.23: Possible scenarios in the extended protocol, consumer side.

ends. If, at a later point in time, the provider agent decides to drop its commitment in favor of other negotiations, it will indicate this in the acceptance phase, by returning an empty agreement document to the consumer agent, and paying the agreed upon penalty. If it becomes clear during the acceptance phase that the consumer agent has dropped its commitment, the provider agent will receive payment. Figure 4.24 shows the possible scenarios for the provider agent.

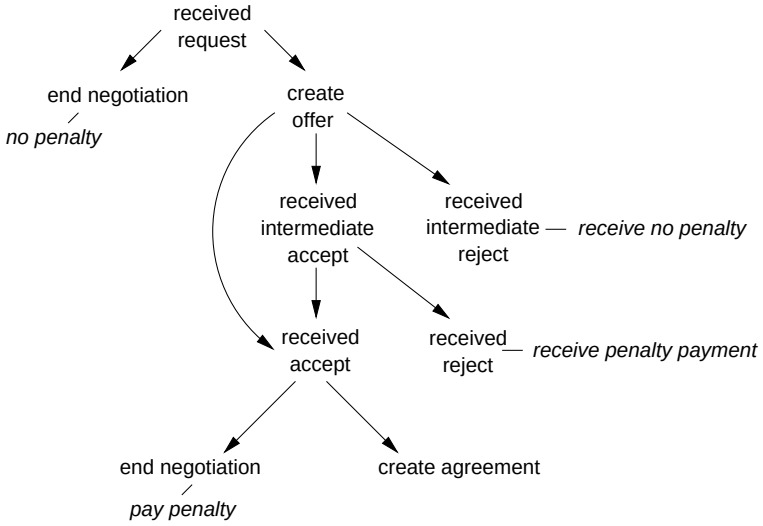

Figure 4.24: Possible scenarios in the extended protocol, provider side.

4.4.5 Scenario D Example Trace

To demonstrate the extension of the framework, two example scenarios are presented in this section: Scenario D1 presents a situation in which one energy consumer agent negotiates with two energy provider agents, and drops its commitment to one of the provider agents; scenario D2 presents a situation in which one energy provider agent negotiates with two energy consumer agents, and drops its commitment to one of the consumer agents. The scenarios demonstrate how both consumers and providers can drop commitments when new and better negotiation options present themselves. For simplicity, it is assumed that the mediators in this scenario only fulfill a passive intermediary role, passing through the received negotiation messages, and not influencing the negotiation process. These interactions are not explicitly specified.

Scenario D1

In this scenario, an energy consumer agent first engages in negotiations with energy provider agent A: The template is requested from the provider. The template indicates the maximum amount of energy that can be requested (750 units), and the penalty that will be imposed if any negotiations based on this template are ended by the consumer agent, without establishing an agreement (Pprov:15). The consumer agent examines the template, and creates a negotiation request for 500 energy units. The request contains the energy amount requested, the original penalty as indicated by the provider agent, and the penalty set by the consumer agent, which is imposed if the provider agent ends negotiations prematurely (Pcons, 5).

Energy provider agent A receives the request, and decides to accept the request and create an offer. The offer is returned to the consumer agents, and includes the price set by the energy provider agent (price: 200). Subsequently, the energy consumer agent decides to commit to the offer by issuing an intermediate accept. From this point, if either energy provider agent A or the energy consumer agent decides to withdraw from the negotiation, a penalty will have to be paid to do so.

The energy consumer agent decides to attempt to negotiate with another energy provider agent, provider agent B. The template specified by this provider agent indicates that it can also supply the requested amount of energy. The consumer agent issues a negotiation request to this provider, to request the price this provider agent asks for 500 units. The provider agent responds with an offer specifying a price of 180, and a penalty of 5, if negotiations are ended prematurely. The consumer agent receives the offer, and decides which offer is to be accepted, and which is to be rejected. Rejecting the old offer will result in a penalty of 15, but the difference in price of both offers makes it worth while to make this choice. The consumer agent first accepts the new offer, to ensure that provider agent B has not yet decided to end negotiations, after which it informs provider agent A of rejection, and paying the agreed upon penalty. Figure 4.25 shows the interactions taking place in this scenario between the negotiation participants.

Scenario D2

In this scenario, an energy provider agent enters into negotiations with two consumer agents, and drops commitment to one negotiation, favoring the most profitable negotia-

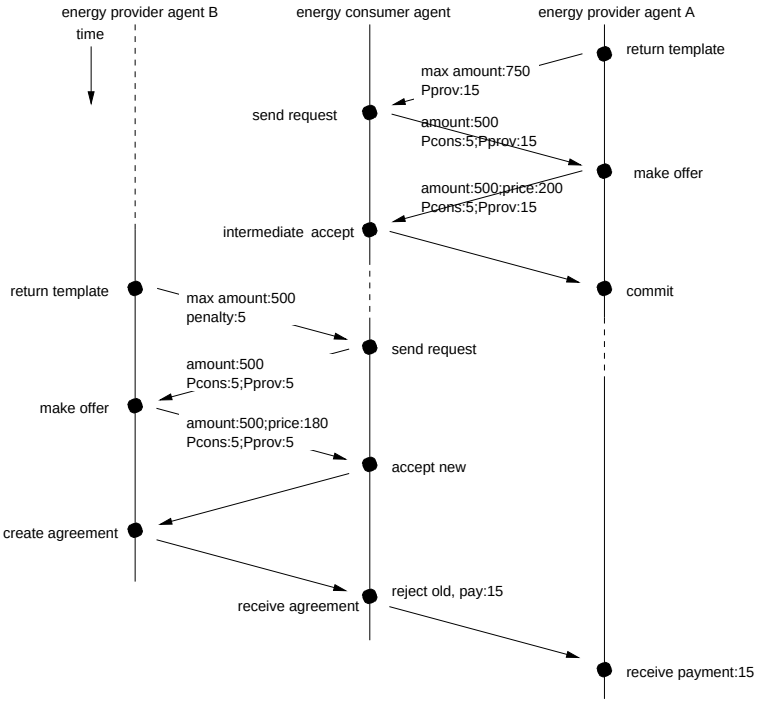

Figure 4.25: Scenario D1

tion. First, the provider agent is contacted by energy consumer agent B, with a request for the template. The template is returned to the consumer agent, indicating that the maximum amount of energy that can be request is 750 units, and the penalty is set to 15. Consumer agent B decides to issue a negotiation request for 500 energy units, and sets the decommitment penalty to 5. The provider agent decides to accept the request, and commits to it by creating an offer, with a price of 200. The offer is accepted by the consumer agent. From this point, a withdrawal of either of the two agents from the negotiation now results in a penalty payment. Figure 4.26 shows the interactions taking place in scenario D2.

Subsequently, the energy provider agent is contacted by consumer agent A, which also requests the template, and creates a request for 500 energy units, and a penalty of 10. The provider agent decides to create an offer in response to the request, with a price of 210. The offer is accepted (immediately, no intermediate acceptance by the consumer in this scenario) by consumer agent A, which means that the provider can now decommit from the negotiation with consumer agent B, and pay the penalty of 5. When consumer agent B indicates that it wants to accept the agreement, the provider agent returns an empty agreement, indicating it has ended this negotiation, and will pay the agreed upon penalty.

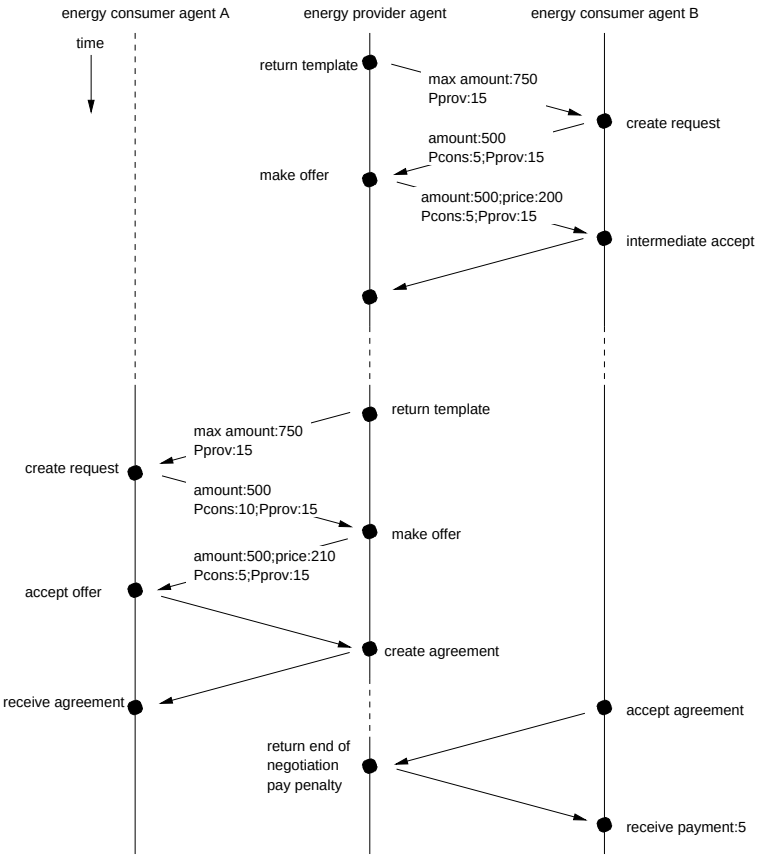

Figure 4.26: Scenario D2

4.4.6 Scenario Discussion

The framework extension presented in this scenario allows for both negotiating parties to indicate levels of commitment during negotiations by settings decommitment penalties, and dropping commitments during negotiations in exchange for payment of these penalties. The extension implements a sequential decommitment mechanism, in which provider agents always move last, as they await the acceptance/rejection of the agreement from the consumer agent. Penalties are not fixed between negotiations, but are established per negotiation, using the negotiation templates, and negotiation request documents.

The extensions to the framework that are required for decommitment are relatively straightforward: An additional section has been defined in the negotiation documents, and an additional intermediate acceptance phase is inserted in the protocol, which uses the same interactions (accept/reject) already present within the original protocol, but with different semantics.

4.5 Summary and Conclusions

The energy negotiation model as presented in this chapter enables energy providers to form virtual organizations, represented by mediator agents. Forming organizations has the advantages that (i) energy providers can offer their energy capacity collectively, and that (ii) organizational policies concerning the use of energy resources can be applied. These policies can ensure that negotiations within the provider organization are performed effectively: The mediator can intelligently choose the providers with which to negotiate, to achieve higher-level organizational goals, such as reducing negotiation times, or specific energy allocation goals (fair division among energy resource providers, prioritize consumer agents based on their identities, etc.).

The applicability to the energy management domain, and the flexibility of the framework has been demonstrated: Using our framework, energy negotiations are modeled as a succession of negotiations for *agreements*, where each agreement describes the amount of energy that is to be delivered, the reliability of the delivered energy, price, and the duration of delivery that has been agreed upon.

The framework has been instantiated, and negotiation policies have been defined for the negotiation participants that enable competition between energy providers, and between groups of energy providers. The simulations that have been performed show that the framework can be used to model different negotiation types in the domain of distributed energy management.

Not all negotiation models are encompassed by the negotiation language and interaction protocol specified in negotiation framework. The framework's potential to model other negotiation types is shown through two example scenarios, in which extensions are presented of the basic framework components: An auctioning model extension, and an extension to enable support for decommitment penalties are presented, which demonstrate the extension possibilities of the negotiation framework. The negotiation scenarios cannot easily be compared to each other: In the auctioning scenario, consumers compete with each other, which can be perceived as a disadvantage from a consumer point of view, but as an advantage for providers. In the decommitment scenario the situation is reversed, as providers compete with each other in the negotiation process. Choosing between these negotiation approaches with the goal of modeling a negotiation process will be largely determined by domain-specific factors such as the relative sizes of the consumer an providers populations, and the availability/scarcity of services for negotiation.

4.5.1 Future Work

From a performance perspective, the negotiation model needs to be further examined. The basic negotiation protocol as described in Section 4.2 has features that can contribute to the performance of negotiations: Advertisements are used to initiate negotiations, limiting the allowed negotiation space and negotiation candidates. This can reduce overall negotiation times, and improve the negotiation results obtained within this time. Furthermore, the protocol is based on a single negotiation cycle, which prevents lengthy negotiation interactions. In realistic settings however, re-negotiations will be necessary when negotiation results are not considered acceptable by energy consumer agents.

Also, the energy negotiation model as described in this chapter focuses primarily on the role of the mediator agent in the negotiation process. The model can be further extended to include additional elements specific to the energy domain, such as pricing models, dynamic energy provider and consumer agent populations, and multiple virtual organizations.

Finally, the energy negotiation model as presented in this chapter can be refined to model different types of realistic energy resources, providers, and consumers more closely.

Chapter 5

Multi-Agent System Resource Management: Resource Negotiation in AgentScape

Next generation agent platforms will consist of large-scale, distributed middleware layers, providing support for many agent applications simultaneously. Mobile agents in these environments need to share limited platform resources with many other agents. These resources are subject to access and usage policies, set by owners and system administrators. Facilities need to be provided by agent platforms, enabling agents to obtain guarantees concerning service access, while simultaneously ensuring that resource access and usage policies are not violated.

This chapter describes the implementation of the negotiation framework in the AgentScape platform [96]. The aim of this chapter is to evaluate the applicability of the framework in the domain of distributed agent middleware.

In AgentScape, agents can enter into negotiations with mediators representing groups of hosts, and establish agreements describing service access and usage conditions. Prior to migrating to a new AgentScape platform, agents need to:

- locate resources;
- determine whether the required resources can be obtained at reasonable cost;
- negotiate guaranteed access to these resources.

These resources can be low-level computational resources such as CPU-time, working memory, communication bandwidth, or higher-level services such as web service access or database access. For example, an agent which has been written in Java requires that the platform to which it migrates offers a Java run-time environment.

Section 5.1 introduces the AgentScape middleware. Section 5.2 describes the mapping of the negotiation framework onto the AgentScape architecture. Sections 5.3, 5.4, and 5.5 discuss the implementations of the consumer, mediator, and provider agents in

the AgentScape middleware in more detail. Section 5.6 presents a number of negotiation experiments. Section 5.7 discusses the implementation of the framework. Section 5.8 summarizes the main points of this chapter. Section 5.9 presents the final conclusions.

5.1 Introduction: AgentScape

AgentScape is a distributed middleware platform designed to support large-scale distributed agent applications. The goal of AgentScape is to realize a secure, scalable, and fault-tolerant middleware layer for agent applications. Furthermore, AgentScape is designed to be an *open* system: Supporting multiple code bases and offering interoperability with other agent platforms.

Agents in AgentScape

AgentScape provides support for migration and run-time support of agents: AgentScape implements an agent life cycle model defining a number of states and state transitions. This model (see Figure 5.1) consists of four states and related transitions:

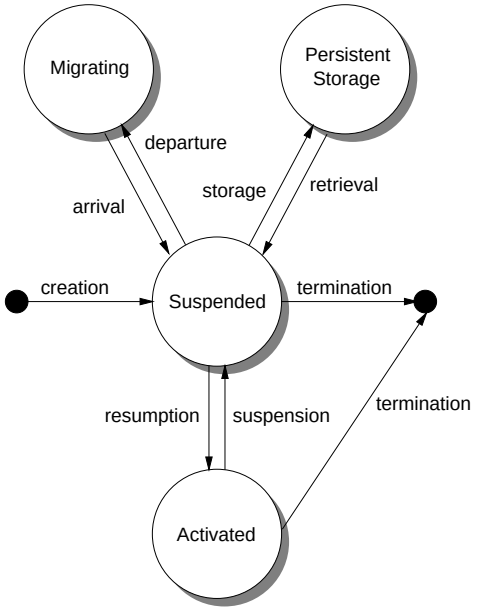

Figure 5.1: AgentScape life cycle model.

- The *active* state, in which the agent is running and able to communicate with its environment and pursue its goals;
- the *suspended* state, in which an agent is temporarily stopped and unable to communicate, but not removed from the middleware instance. The *state* of the agent

is stored within the middleware (i.e. in an *agent container*, see the next section for more details). Depending on the policies of the middleware, incoming messages addressed to the agent may be buffered for later delivery, or not delivered;

- the *migrating* state, during which the agent is being moved to another host within the middleware. Depending on the policies of the middleware, incoming messages may be buffered and forwarded to the new location of the agent, or not delivered;

- the *persistent storage* state, in which an agent is removed from the middleware and stored in persistent storage outside the middleware. Incoming messages are not delivered.

For communication purposes, AgentScape implements a naming service, and simple low-level message-based communication primitives for transparently and securely routing messages to and from agents. AgentScape also provides higher-level functionality for agents to discover other agents and/or services. These are not further described in this thesis.

Access to web services in AgentScape is offered to agents through a web service gateway, giving agents the ability to communicate with web services using the SOAP/XML protocol [1]. The gateway acts as a proxy between the web service and an agent wishing to access the service, by redirecting the SOAP traffic from the agent to the gateway instead of directly to the web service, allowing AgentScape to monitor and control agent access to web services.

AgentScape Middleware Architecture

The AgentScape middleware provides agents with a world consisting of *locations*. Each AgentScape location provides agents with a run-time environment. AgentScape locations are supported by one or more hosts. Each host in an AgentScape location runs a *Host Manager*, and one or more *Agent Servers*. Each location has its own *Location Manager*, which is responsible for managing the host managers within its location. Each host manager within a location is responsible for registering its host with the location manager. Host managers only communicate with other host managers (its peers), or with the location manager of its location. Locations are connected to each other by their location managers. This configuration is in fact a super-peer architecture. Figure 5.2 depicts the entities within an AgentScape location.

Figure 5.3 presents an overview of the AgentScape architecture from the perspective of a single host. Three layers are distinguished within the middleware: A *kernel* layer, providing basic support functionality for the *middleware layer*. The middleware layer implements the functionality required to support mobile agent applications and agent communication. Middleware layers on separate hosts communicate with each other through the kernel layer using an RPC-mechanism. The *application layer* is the layer within which the actual agent applications are run. This layer also provides additional higher level application support (e.g. agent directory services).

Agents in AgentScape are not aware of the distributed and modular nature of the middleware. Agents only see a single entity representing the middleware: the *agent server*.

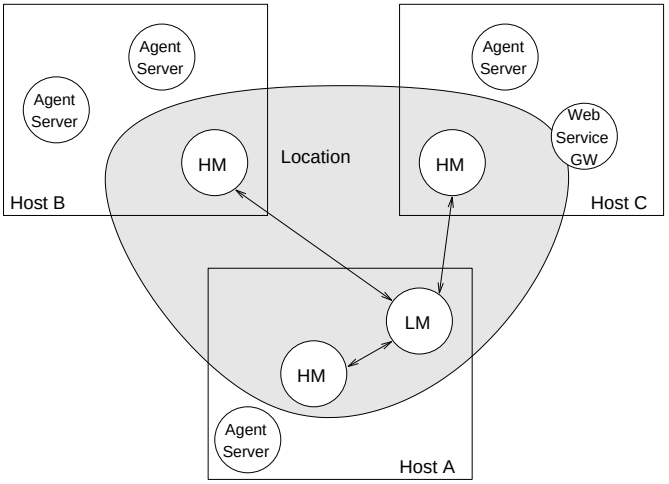

Figure 5.2: An AgentScape location.

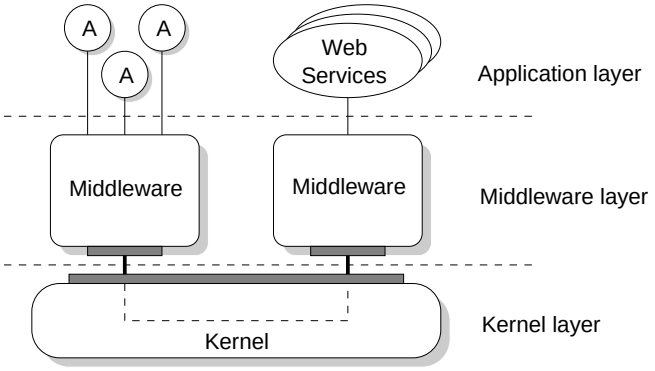

Figure 5.3: AgentScape middleware.

This agent server presents agents with an *agent interface*, through which agents communicate with other agents, interact with services, and migrate to other locations. Furthermore, agents in AgentScape are not aware of the individual hosts supporting the middleware. From the agent perspective, locations are the most basic elements in an AgentScape world.

Agent communication within AgentScape is message-based: Agents formulate a message and rely on the AgentScape middleware to route the message to the correct location and deliver the message to the destination agent. As an open agent platform, AgentScape supports only low-level secure communication. Higher-level message abstractions (i.e. *Agent Communication Languages*) are the responsibility of the application layer.

AgentScape supports weak agent mobility: Agents are not bound to the run-time environment of the location at which they were first started. Agents can leave an AgentScape location and move to another location (e.g. to use resources available at the target loca-

tion). AgentScape agents can also carry data from one location to the next, using the storage space available in a so-called *agent container* [93]. An agent container is a data structure for storing agent code, state data, and miscellaneous data. As shown earlier in the AgentScape life cycle model, agents are first suspended before being moved to another location, which results in the state of the agent being stored in the agent container associated with the agent. Once the suspension process is completed successfully, the agent container containing the agent and its associated data are moved to the destination location.

The kernel has been designed as a separate entity from the AgentScape middleware, resulting in a robust and scalable layer, with a well-defined interface which is available for use through an RPC mechanism by the middleware. The kernel provides three types of functionality:

- Secure communication: The kernel allows middleware on separate hosts to communicate with each other through encrypted communication channels. This allows the middleware to exchange data (including encapsulated agent communication data) in a secure way.
- Secure agent container management: Within the AgentScape middleware, agent code and data are stored in *agent containers*. To enable management of agent containers, the kernel provides a number of calls to create, modify, and delete agent containers. Agent containers are stored using encryption, preventing unauthorized access to the agent code and data stored within.
- Secure migration: The kernel supports secure migration of agent containers from one kernel to another (most often from a host in one location to a host in another location). Security mechanisms enable the kernel to detect tampering with agent containers during transfer.

5.2 AgentScape Negotiation Architecture

This section describes the application of the negotiation framework to the AgentScape middleware. For each of the agents identified in the negotiation framework (consumer, mediator, and service provider agents), it is determined how the agent is instantiated in the middleware architecture. Furthermore, the services are identified that are the topics of negotiation.

5.2.1 Agentscape Framework Implementation

The negotiation architecture is integrated into the AgentScape architecture, with the goal of enabling system adminstrators (i.e. administrators of AgentScape locations or individual hosts) to define resource access policies for the resources within their administrative scope, and to allow agents to negotiate for access to these resources. This section describes the mapping of the framework elements to components of the AgentScape architecture, and describes the additions and modifications that are made to the AgentScape middleware to accomplish this.

In AgentScape, agents implement the *consumer agent* role within the negotiation process. Agents negotiate with AgentScape *locations* to obtain access to services offered by these locations. Location managers act as *mediator agents*, on behalf of an AgentScape location. Within an AgentScape location, host managers act as *service provider agents* representing specific resources. An AgentScape location represents a virtual organization of host managers. Location managers represent locations in the negotiation process: They negotiate with host managers in their location, on behalf of agents. Figure 5.4 shows the mapping of the negotiation framework onto the AgentScape middleware.

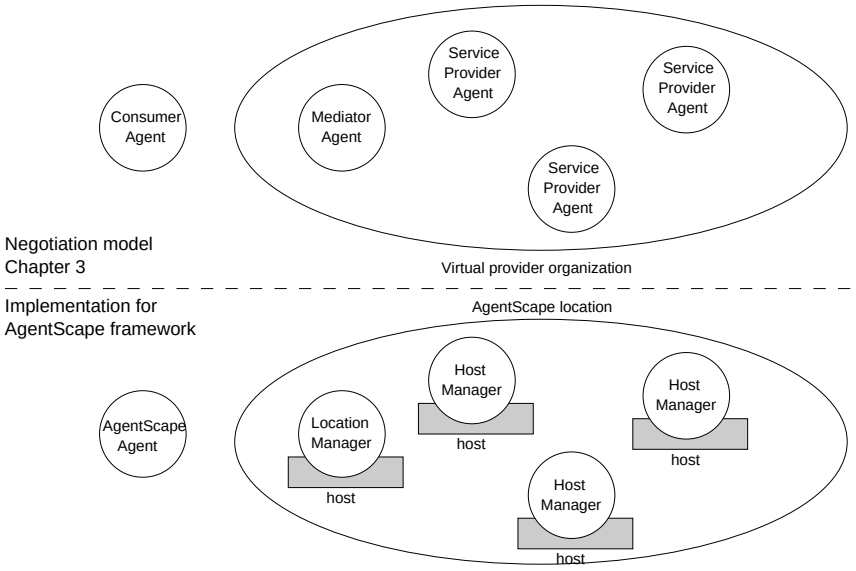

Figure 5.4: Mapping of the negotiation framework elements onto the AgentScape middleware.

The AgentScape prototype is extended with a Java implementation of the negotiation framework. An existing AgentScape kernel implementation developed as part of the ongoing AgentScape project provides the required kernel layer services (i.e. secure communication and migration), and was used without modifications. An existing Agent Server which is available within the AgentScape prototype framework was modified to allow for monitoring and control of resource usage by the agents it supports, including monitoring of CPU-time consumption. Similarly, existing Host Manager and Location Manager prototype implementations were extended to include the components required for the negotiation architecture, and the additional interface elements required for negotiation protocol communication. The original AgentScape agent interface that is used by agents to communicate with the world and the middleware is extended to include the required negotiation protocol calls. The existing migration protocol as implemented in the AgentScape prototype was extended to include the negotiation phase: Prior to migration, agents are required to obtain an agreement with the location that is the target of migration. The identifier of the agreement is then used to initiate the migration process.

Furthermore, an existing example WS-Agreement XML-Schema specification pro-

vided by the Open Grid Forum was used as a basis to generate the negotiation documents that are exchanged during the negotiation process. Additionally, an XML-Schema was created defining the negotiable elements (i.e. AgentScape middleware resources) that can be expressed in the negotiation documents.

The negotiation framework explicitly does not specify (domain-specific) negotiation policies, or a policy specification language, as this is considered to be domain-specific. In the current version of AgentScape, no specific policy language has been defined to express policies on a more abstract level. Instead, policies are specified directly in the agents, location manager, and host manager implementations. This makes it difficult for other parties to add or modify negotiation policies at the current time, as they have to apply the modifications directly in the AgentScape source code. Selecting and/or defining a suitable language to specify negotiation policies that can be configured and instantiated at run-time is the subject of future research.

In the following sections, further details of the integration of the negotiation architecture in the AgentScape framework are discussed.

5.2.2 Topics of Negotiation

In AgentScape, the topics of negotiation (i.e. the *services* in the terminology of the negotiation framework) are middleware resources. Currently, three resource descriptions are specified: *Agent Run-time Environments*, *CPU-time*, and *Web Service Access*. The structure of these resource descriptions are specified in the XML-Schema language. The specifications can be extended to include additional resources when needed, such as memory usage, or disk usage, etc. Currently however, the implementation only supports negotiation about the three above-mentioned resources. Each of these resources are described in more detail in the following sections.

```
<xs:element name="EnvironmentResource"
        type="agentscape:EnvironmentResourceType"/>
<xs:complexType name="EnvironmentResourceType">
 <xs:sequence>
  <xs:element ref="agentscape:Language" minOccurs="1" maxOccurs="1"/>
 </xs:sequence>
</xs:complexType>

<xs:element name="Language" type="agentscape:LanguageType"/>

<xs:complexType name="LanguageType">
 <xs:simpleContent>
  <xs:extension base="xs:string">
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
```

Example 5.1: Specification of the run-time environment resource description.

Agent Run-time Environment

In AgentScape, run-time environments for agents are provided by *agent servers*. Each agent server provides a language dependent run-time environment. The AgentScape middleware implementation currently provides two types of agent servers, one offering a Java and one offering a C run-time environment. Agents can negotiate for specific run-time environments with the AgentScape middleware using resource descriptions based on the specification as shown in Example 5.1.

The specification defines a top-level *EnvironmentResource* element, containing a single *Language* element. This element can be used in negotiations to indicate run-time environments, as shown in the example below. The specification in its current form is overly simplified: Future versions of the specification should separate the various aspects defining a run-time environment, such as *language name*, *version* and *dialect*, instead of using a single value representation.

Administrators of AgentScape hosts can specify in negotiation advertisements which run-time environments they have available. Example 5.2 depicts an advertisement specifying three available run-time environments: *Java-1.4.2*, *Java-1.5.0*, *C/C++*.

```
<Template>
 ...
  <ServiceDescriptionTerm serviceName="RuntimeEnvironment">
   <EnvironmentResource>
    <Language/>
   </EnvironmentResource>
  </ServiceDescriptionTerm>
 ...
 <CreationConstraints>
  <Item>
   <Location>
    //ServiceDescriptionTerm[@serviceName='RuntimeEnvironment']
          /EnvironmentResource/Language
   </Location>
   <Restriction type="enumeration">
    <enum>Java-1.4.2</enum>
    <enum>Java-1.5.0</enum>
    <enum>C/C++</enum>
   </Restriction>
  </Item>
 </CreationConstraints>
</Template>
```

Example 5.2: An advertisement containing the *EnvironmentResource* resource description.

A negotiation request based on this advertisement, in which an agent requests a *Java-1.4.2* run-time environment, is depicted in Example 5.3.

CPU-time

Agents can also negotiate with AgentScape platforms for required CPU-time by using resource descriptions based on the specification shown in Example 5.4.

The specification is used by administrators of AgentScape hosts to specify negotiation advertisements containing constraints on the amount of CPU-time that agents may request. The current specification allows for the expression of CPU-time requirements in

```
<Agreement>
...
 <All>
  <EnvironmentResource>
   <Language>Java-1.4.2</Language>
  </EnvironmentResource>
 </All>
...
</Agreement>
```

Example 5.3: A negotiation request containing the *EnvironmentResource* resource description.

```
<xs:element name="CPU-timeResource"
        type="agentscape:CPU-timeResourceType"/>

<xs:complexType name="CPU-timeResourceType">
 <xs:sequence>
  <xs:element name="CPU-time" type="xs:positiveInteger"/>
 </xs:sequence>
</xs:complexType>
```

Example 5.4: Specification of the CPU-time resource description.

milliseconds, and does not account for differences in CPU type or speed. Example 5.5 depicts an example of an advertisement, in which the maximum CPU-time that may be requested is limited to 20000 milliseconds.

```
<Template>
 ...
 <ServiceDescriptionTerm serviceName="CPU">
  <CPU-timeResource>
   <CPU-time/>
 </CPU-timeResource>
 </ServiceDescriptionTerm>
 ...
 <CreationConstraints>
  <Item>
   <Location>
    //ServiceDescriptionTerm[@serviceName='CPU']
            /CPU-timeResource/CPU-time
   </Location>
   <Restriction type="maxValue">
    <maxValue>20000</maxValue>
   </Restriction>
  </Item>
 </CreationConstraints>
</Template>
```

Example 5.5: An advertisement containing the *CPU-timeResource* resource description.

An example of a negotiation request created by an agent based on this advertisement (in which an agent requests 10000 milliseconds CPU-time), is depicted in Example 5.6.

Web Service Access

Agents negotiate for access to web services by using the resource descriptions based on the specification as shown in Example 5.7. Successful negotiations result in the web

```
<Agreement>
 ...
 <All>
  <CPU-timeResource>
   <CPU-time>10000</CPU-time>
  </CPU-timeResource>
 </All>
 ...
</Agreement>
```

Example 5.6: A negotiation request containing the *CPU-timeResource* resource description.

service gateway allowing access to the web services requested by the agent.

```
<xs:element name="WebServiceResource"
        type="agentscape:WebServiceResourceType"/>
<xs:complexType name="WebServiceResourceType">
 <xs:sequence>
  <xs:element name="WebServiceName" type="xs:string" minOccurs="1"
                                  maxOccurs="unbounded"/>
 </xs:sequence>
</xs:complexType>
```

Example 5.7: Specification of the *WebServiceResource* resource description.

AgentScape administrators can use these resource descriptions in negotiation advertisements to specify web services that may be accessed. For any web services to which access is allowed through the gateway, the names are specified in the advertisement. Example 5.8 depicts an example of an advertisement containing a web service resource description with two web services (*WebServiceA*,*WebServiceB*).

```
<Template>
 ...
 <ServiceDescriptionTerm serviceName="WebService">
  <WebServiceResource>
   <WebServiceName/>
  </WebServiceResource>
 </ServiceDescriptionTerm>
 ...
 <CreationConstraints>
  <Item>
   <Location>
    //ServiceDescriptionTerm[@serviceName='WebService']
            /WebServiceResource/WebServiceName
   </Location>
   <Restriction type="enumeration">
    <enum>WebServiceA</enum>
    <enum>WebServiceB</enum>
   </Restriction>
  </Item>
 </CreationConstraints>
</Template>
```

Example 5.8: An advertisement containing the *WebServiceResource* resource description.

A negotiation request based on this advertisement, in which access to one of the web services (*WebServiceB*) is requested by an agent, is shown in Example 5.9.

```
<Agreement>
 ...
  <All>
   <WebServiceResource>
    <WebServiceName>WebServiceB</WebServiceName>
   </WebServiceResource>
  </All>
 ...
</Agreement>
```

Example 5.9: A negotiation request containing the *WebServiceResource* resource description.

The AgentScape resource descriptions introduced above allow for negotiation about three different AgentScape resource types: *Agent Run-time Environment*, *CPU-time*, and *Web Service Access*. To support the actual implementation of established agreements concerning these resources, the AgentScape middleware provides monitoring and control facilities for each of these resources. Other AgentScape resource types can be added to the negotiation language, given that the AgentScape middleware provides the required monitoring and control functionality. For more information on resource monitoring and control in AgentScape, see Section 5.5.

In the following sections, the consumer agent, mediator agent, and service provider agent entities of the negotiation framework are mapped onto the AgentScape middleware.

5.3 Consumer Agent

In AgentScape, agents fulfill the *consumer agent* role in the negotiation framework. Agents migrate from one location to another to access resources. Agents negotiate resource access prior to migration: An agent first determines its resource requirements (based on its current goals and tasks), and subsequently negotiates with one or more target locations, to determine which location can provide the best offer. When a suitable offer has been received, the agent accepts the offer, after which it migrates to that location to claim the resources. Figure 5.5 depicts the steps involved in the migration process.

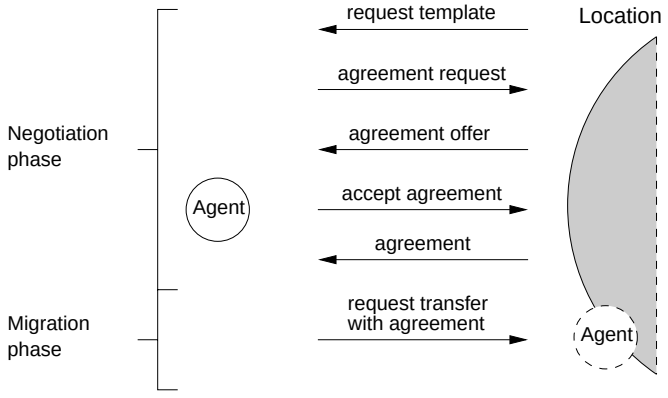

Figure 5.5: Migration using the negotiation framework.

This section describes the negotiation functionality implemented in AgentScape. Details are presented for each of the three main negotiation tasks: *Advertisement management*, *Request management*, and *Agreement management*.

5.3.1 Advertisement Management

Prior to negotiations, agents in AgentScape need to determine which locations are suitable candidates for negotiation. The AgentScape middleware offers a *directory service* for this purpose. The details of this service are beyond the scope of this thesis. Once a list of candidate locations has been obtained by an agent, advertisements are requested using the following call:

- *TemplateList requestTemplates(locationID)*
  This call allows agents to request the available advertisements from a location, specified by the *location identifier* argument.

In the negotiation framework described in Chapter 3, consumer agents specify a list of *service interests* when requesting advertisements from mediator agents. Support for this is, however, not included in the current implementation of the framework within AgentScape. Agents that request advertisements using the above call simply receive all available advertisements from the target location. As a result of this call, an agent receives a list of advertisements, in the form of advertisement documents. These documents specify the resources available at the location, and any negotiation constraints that have been defined over these resources (e.g. see Example 5.8).

5.3.2 Request Management

After the advertisements have been obtained, an agent formulates a negotiation request. The contents of this request are based on (i) the agent's resource requirements, and (ii) the advertisements. As an example, consider an agent that has the following resource requirements:

- *A:* Access to a Java-1.4.2 run-time environment.
- *B:* Access to one of two web services: *WebServiceA* or *WebServiceB*.

Example 5.10 shows the requirements translated into a negotiation request. Note the use of the term compositors *All* and *ExactlyOne* to indicate the relationships between the resource requirements.

The following call is used to communicate the request to the target location:

- *AgreementOffer requestAgreement(locationID, agreementRequest)*
  Using this call, agents send an agreement request document to a location.

```
<Agreement>
 ...
 <ExactlyOne>
 <All>
  <EnvironmentResource>
   <Language>java-1.4.2</Language>
  </EnvironmentResource>
  <ExactlyOne>
   <WebServiceResource>
    <WebServiceName>WebServiceA</WebServiceName>
   </WebServiceResource>
   <WebServiceResource>
    <WebServiceName>WebServiceB</WebServiceName>
   </WebServiceResource>
  </ExactlyOne>
 </All>
 </ExactlyOne>
 ...
</Agreement>
```

Example 5.10: An AgentScape negotiation request.

5.3.3 Agreement Management

In response to a request an agent receives one or more negotiation offers from the different locations. The agent selects the best offer. The selection process is governed by negotiation strategies specific to the agent application. The AgentScape implementation of the framework does not provide support for negotiation strategies for consumer agents: It is left up to agent developers to implement strategies to compare and select negotiation offers.

After a selection has been made, the agent indicates acceptance of the offer using the following call:

- *Agreement acceptAgreement(locationID, agreementOffer)*
  The agent needs to supply the identifier of the location which made the offer, and the agreement offer that the agent accepts. The call returns the final agreement document specifying all negotiated resource specifications.

Finally, the agent migrates to the target location using the *move* call. Before the actual migration process is initiated, the target location needs to be informed about the agreement the agent wishes to use at that target location. The information in the agreement may affect the placement of the agent within the location (e.g. if a specific run-time environment was negotiated by the agent, the migration process must ensure that the agent is placed on the host offering this run-time environment). For this purpose, the original AgentScape *move(locationID)* call used by agents to request migration to another location, has been extended to include the identifier of the agreement that it has established with the target location:

- *move(locationID, agreementID)*
  This call informs the middleware that the agent wishes to migrate to a different location, using the agreement indicated by *agreementID*.

5.4 Mediator Agent

In AgentScape, a location manager (LM) is the primary entity which manages a location: It maintains information on the hosts within its location, and is the contact point for agent migration both to and from the location.

The location manager implements the mediator role in the negotiation implementation: Mediating negotiation processes between service providers (i.e. host managers) within the location, and agents (i.e. agents outside the location requesting migration, or agents within the location requesting new agreements). Location-wide negotiation policies influence the negotiation interactions, and resulting resource access. For example, a location manager may differentiate between agents issuing negotiation requests and provide agents with different negotiation options based on the identity of the agent (e.g. giving well-known agents or agent 'owners' a better deal).

The AgentScape location manager itself consists of a number of *modules* implementing functionality related to the various location management tasks, such as managing agent migration to and from a location, and managing the configuration of a location. Figure 5.6 depicts the modules within the location manager. The *Location Management Module* (LMM) of the LM maintains information on registered HMs (e.g. middleware contact addresses, etc.). The *Migration Management Module* (MMM) manages incoming and outgoing agent migrations. The location manager is extended with the *Negotiation Module* (NM), implementing the main negotiation subtasks of the mediator architecture: *Advertisement Management*, *Request Management*, and *Agreement Management*. The next section discusses these three subtasks.

5.4.1 Advertisement Management

The Advertisement Management (AM) component of the negotiation module periodically queries the LMM to determine if changes have occurred in the configuration of the location. AgentScape locations are dynamic entities: Hosts can join or leave a location at any point in time. This also affects the negotiation infrastructure of the location. The currently implemented policy regarding management of advertisements, is the following: If a host is unresponsive for a certain period, its advertisements are removed; When a new host joins an existing location, its advertisements are requested when needed, and incorporated into the location managers combined advertisement.
To request advertisements from a host manager, a location manager uses the following call:

- *TemplateList requestTemplates()* This call of the host manager middleware interface allows a location manager to request available advertisements.

Combining Advertisements

Combining advertisements consists of combining resource description terms and negotiation constraints, as described earlier in Chapter 3. The combination process is governed by (i) domain-specific knowledge concerning resource combinations, and (ii) combination policies defined at the organizational level, prescribing (dis)allowed resource com-

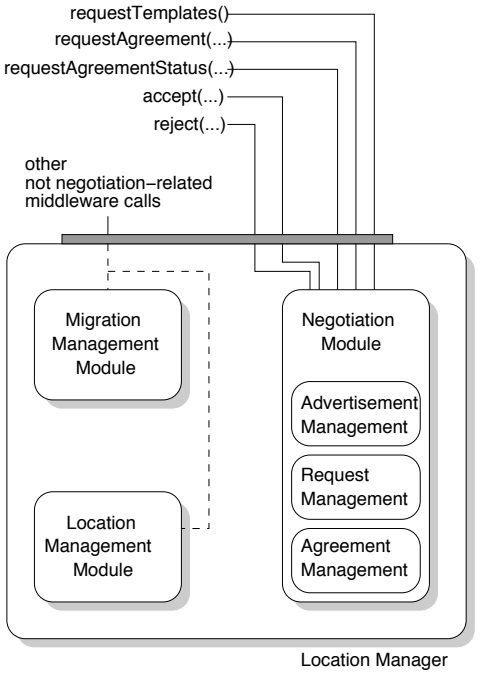

Figure 5.6: Location manager modules.

binations. Below, the constraint combination process in the AgentScape domain is discussed: Three examples of the advertisement combination process are given. Example 1 is a straightforward combination process. Example 2 shows the role of domain-specific resource combination knowledge. Example 3 shows the role of combination policies.

5.4.2 Combining Advertisements: Example 1

In this example, two host manager advertisements are combined by a location manager into a combined advertisement. Consider the following two host manager advertisements shown in Examples 5.11 and 5.12.

The two host manager advertisements both offer similar resources, but with different constraints: Host manager A indicates that two run-time environments are available: *Java-1.4.2*, and *C/C++*. Furthermore, A indicates that it provides access to a web service named *WebServiceA*. Host Manager B indicates that it provides a *Java-1.4.2* and a *Java-1.5.0* run-time environment. B also provides web service access, but to a different web service than A: *WebServiceB*. The AgentScape location manager combines these advertisements using the resource constraint combination operations as described above: Both advertisements offer the same types of services (*RuntimeEnvironment* and *WebService*), no changes are made to the *terms* section. The *Creation Constraints* section is constructed by combining the constraints of both advertisements, as depicted in Example 5.13.

```
...
<OneOrMore>
 <All>
  <ServiceDescriptionTerm serviceName="RuntimeEnvironment">
   <EnvironmentResource><Language/></EnvironmentResource>
  </ServiceDescriptionTerm>
 </All><All>
  <ServiceDescriptionTerm serviceName="WebService">
   <WebServiceResource><WebServiceName/></WebServiceResource>
  </ServiceDescriptionTerm>
 </All>
</OneOrMore>
<Item>
 <Location>
  //ServiceDescriptionTerm[@serviceName='RuntimeEnvironment']/EnvironmentResource/Language
 </Location>
 <Restriction type="enumeration">
  <enum>Java-1.4.2</enum>
  <enum>C/C++</enum>
 </Restriction>
</Item><Item>
 <Location>
  //ServiceDescriptionTerm[@serviceName='WebService']/WebServiceResource/WebServiceName
 </Location>
 <Restriction type="enumeration">
  <enum>WebServiceA</enum>
 </Restriction>
</Item>
...
```

Example 5.11: Advertisement created by host manager A.

```
...
<OneOrMore>
 <All>
  <ServiceDescriptionTerm serviceName="RuntimeEnvironment">
   <EnvironmentResource><Language/></EnvironmentResource>
  </ServiceDescriptionTerm>
 </All><All>
  <ServiceDescriptionTerm serviceName="WebService">
   <WebServiceResource><WebServiceName/></WebServiceResource>
  </ServiceDescriptionTerm>
 </All>
</OneOrMore>
<Item>
 <Location>
  //ServiceDescriptionTerm[@serviceName='RuntimeEnvironment']/EnvironmentResource/Language
 </Location>
 <Restriction type="enumeration">
  <enum>Java-1.4.2</enum>
  <enum>Java-1.5.0</enum>
 </Restriction>
</Item><Item>
 <Location>
  //ServiceDescriptionTerm[@serviceName='WebService']/WebServiceResource/WebServiceName
 </Location>
 <Restriction type="enumeration">
  <enum>WebServiceB</enum>
 </Restriction>
</Item>
...
```

Example 5.12: Advertisement created by host manager B.

```
...
<OneOrMore>
 <All>
  <ServiceDescriptionTerm serviceName="RuntimeEnvironment">
   <EnvironmentResource><Language/></EnvironmentResource>
  </ServiceDescriptionTerm>
 </All><All>
  <ServiceDescriptionTerm serviceName="WebService">
   <WebServiceResource><WebServiceName/></WebServiceResource>
  </ServiceDescriptionTerm>
 </All>
</OneOrMore>
...
<Item>
 <Location>
  //ServiceDescriptionTerm[@serviceName='RuntimeEnvironment']/EnvironmentResource/Language
 </Location>
 <Restriction type="enumeration">
  <enum>java-1.4.2</enum>
  <enum>java-1.5.0</enum>
  <enum>c/c++</enum>
 </Restriction>
</Item><Item>
 <Location>
  //ServiceDescriptionTerm[@serviceName='WebService']/WebServiceResource/WebServiceName
 </Location>
 <Restriction type="enumeration">
  <enum>WebServiceA</enum>
  <enum>WebServiceB</enum>
 </Restriction>
</Item>
...
```

Example 5.13: Combined advertisement created by the location manager.

5.4.3 Combining Advertisements: Example 2

Combining advertisements with multiple service specifications can become complex, especially if term compositors are used to specify service combination restrictions. In this example, schematic representations of advertisements are used, as opposed to the longer XML specifications used in earlier examples.

Figure 5.7 shows an example of the schematic representation of an advertisement: This advertisement specifies two service descriptions, each with value restrictions. The two descriptions are contained within a *ExactlyOne* element (the outer box), which indicates that either the *Run-time environment*, or the *Web Service access* resource may be requested, but not both.

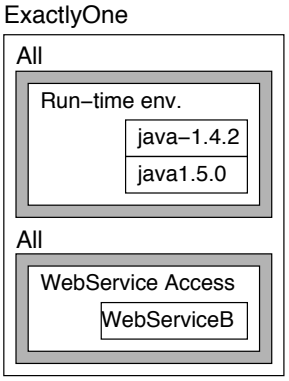

Figure 5.7: Abstract representation of an advertisement.

In this example, the policy implemented by the mediator is to combine the content of the advertisements based on domain-specific resource knowledge: Knowledge describing which resources can or cannot be offered as a combination, due to constraints imposed by the resources. In AgentScape, the *CPU-time* and *run-time environment* resources are strongly related: Both resources are consumed *locally* by agents after migration (i.e. on the AgentScape host on which the agents are placed). The location manager must make sure that agents cannot establish agreements for these resources across multiple hosts (e.g. *CPU-time* on host A, and a *run-time environment* on host B): Agents cannot use both resources simultaneously (e.g. if an agent is placed on host A after migration, the agreed upon *run-time environment* on host B does not make any sense). Consider the host manager advertisements shown in Figure 5.8.

Both advertisements indicate that at the level of individual hosts, either *CPU-time* or *run-time environment* resources can be negotiated separately, or both together. To combine the advertisements, the location manager first generates all combination options: Figure 5.9 shows the combined advertisement.

After this, the location manager determines if the advertisement includes invalid combinations. If so, which is the case in this example, they are removed: Both combinations consist of a *CPU-time* resource on one host, and a *run-time environment* resource on another host. These combinations cannot be used by an agent. Figure 5.10 shows the combined advertisement.

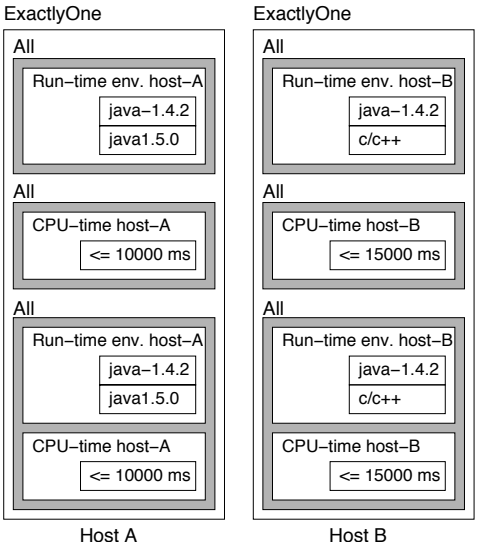

Figure 5.8: Two host manager advertisements containing *CPU-time* and *run-time environment resource descriptions.*

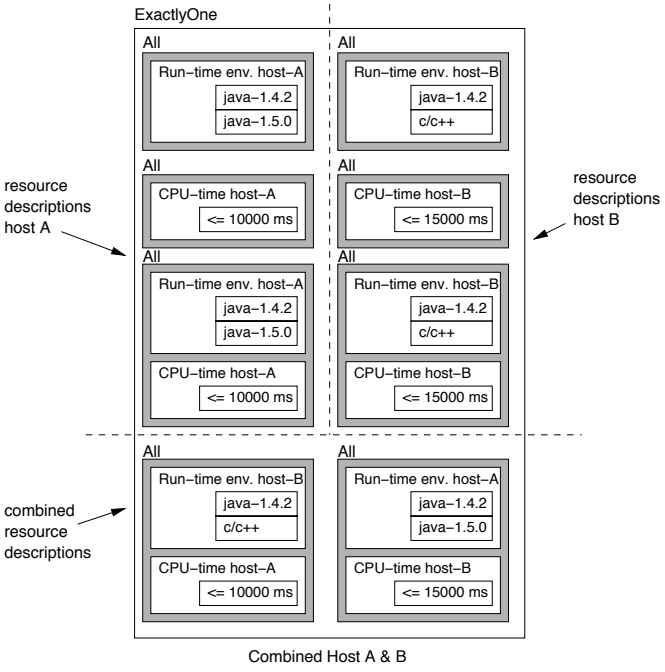

Figure 5.9: The combined advertisement containing all combinations of resources.

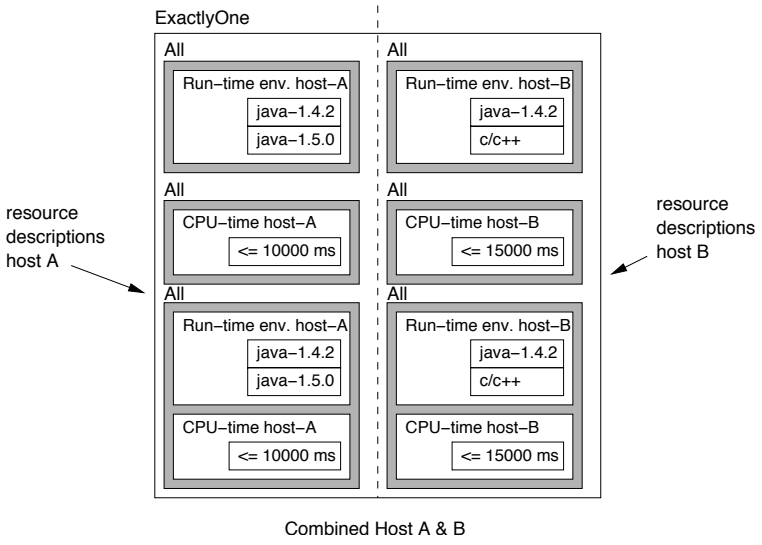

Figure 5.10: The final combined advertisement.

Finally, any combinable service descriptions are combined (as described in the Example 1 above): The individual *CPU-time* and *run-time environment* resource descriptions are combined into single descriptions. The lower sections containing combined *CPU-time* and *run-time environment* descriptions cannot be combined, as the allowed values specified by host A and host B are different: combining these would result in invalid descriptions. For example, it would allow a request for both a *java-1.5.0 run-time environment* and a *CPU-time* value of 15000 ms, which cannot be delivered by either of the two hosts individually. The final combined advertisement is shown in Figure 5.11.

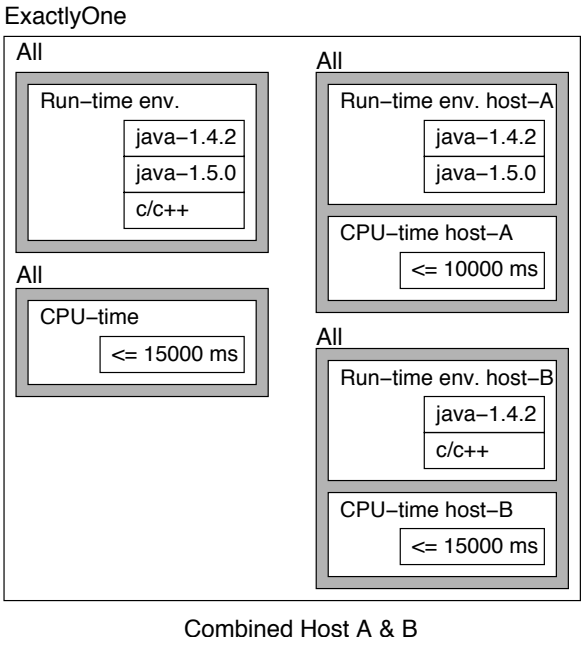

Figure 5.11: The final combined advertisement.

5.4.4 Combining Advertisements: Example 3

In the last example, the role of combination policies in the combination process is shown. Consider the advertisements shown in Figure 5.12.

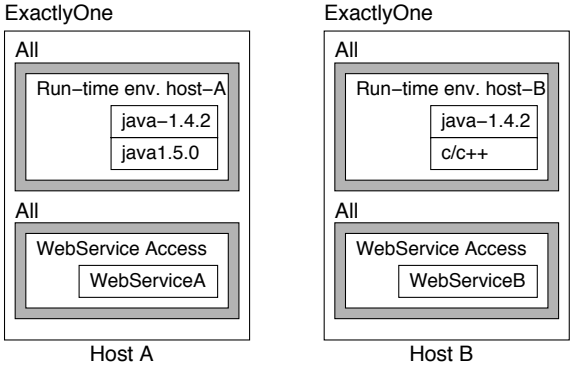

Figure 5.12: Two host manager advertisements containing *web service* and *run-time environment resource descriptions.*

The two advertisements both offer two resources: *Web service access* to a specific web service, and *run-time environments* (WebServiceA,WebServiceB). Both advertisements indicate that at each of the hosts, either one of the resources may be requested, but not both simultaneously.

The location manager first creates a combined advertisement from the two advertisements, containing all combination options. Figure 5.13 depicts the combined advertisement.

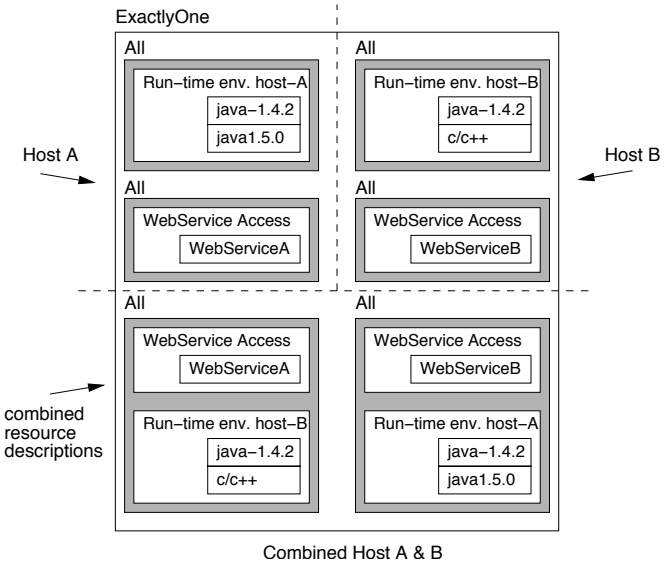

Figure 5.13: The initial combined advertisement based on the advertisements shown in Figure 5.12.

The location manager subsequently applies combination policies to determine if any combinations are present in the advertisement that are not allowed according to these policies. The example combination policy applied in this example is the following: *Resources used by agents may not span multiple hosts*. This policy leads to the removal of the lower two options in the combined advertisement, as these options combine resources from both hosts.

Finally, the location manager combines any resource descriptions that can be combined into single descriptions. In this case the *run-time environment* and *web service access* descriptions can be combined and presented as single options in the advertisement. Figure 5.14 depicts the final advertisement.

5.4.5 Request Management

The main task of the request management component of the location manager is to create negotiation offers in response to negotiation request received from agents. This task is divided into two subtasks:

- **A:** Selecting with which host manager to negotiate, on behalf of the agent;
- **B:** Selecting and combining the received offers from host managers into a negotiation offer for the agent.

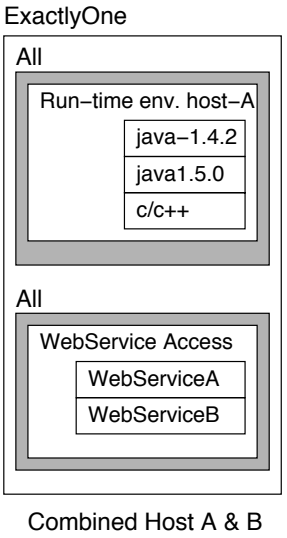

Figure 5.14: The final combined advertisement.

A: Selecting Host Managers

Upon receiving a negotiation request from an agent, the location manager selects the host manager(s) to which it will send a negotiation request. This selection process is governed by three types of knowledge:

1. Host manager advertisements;
2. Domain-specific knowledge about resources and their combinations;
3. Domain-specific negotiation policies.

Below these knowledge types are discussed in more detail, after which an example of the host selection process is given.

1. Host manager advertisements This task determines which hosts advertise the resources requested by the agent. For this purpose, the location manager compares the requested resources in the negotiation request of the agent, to the content of the host manager advertisements. Policies guide which advertisements are selected and combined. For example, a location manager may implement a policy which specifies that at most two advertisements may be combined to limit the number of host managers involved in the negotiation process. In AgentScape, the currently implemented policy does not specify any constraints on the advertisement combination process.

2. Domain-specific knowledge about resources and their combinations. In AgentScape, some resources are consumed locally on the host to which an agent migrates (as

mentioned earlier, e.g. the *CPU-time* resource). However, other resources can be consumed *remotely*. For example, Figure 5.15 shows an example in which an agent resides on host A and uses the *CPU-time* resource, on that host, whilst using the *web service access* resource (i.e. a Web Service Gateway), located on host B.

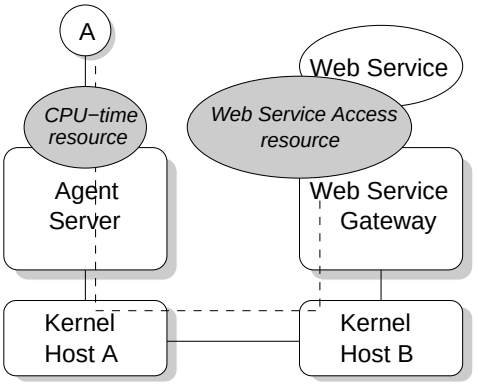

Figure 5.15: An agent using multiple resources on different hosts within a location.

The location manager uses this knowledge about AgentScape resources, together with the advertisement information of the host managers, to decide if a negotiation request will be split into multiple negotiation requests to different hosts.

3. Domain-specific negotiation policies. Negotiation policies influence which host managers are selected as candidates for negotiation. The location manager applies these policies to (i) reduce negotiation overhead, and to (ii) apply organization-wide resource usage strategies.

AgentScape locations can consist of many hosts, each offering multiple resources. An important factor in the negotiation process within AgentScape is the response time: The location manager must provide agents with negotiation offers quickly, to prevent agents from spending too much time waiting, or loosing interest and aborting the negotiation process. Reducing the number of host managers involved in a negotiation sequence reduces the total negotiation overhead, as less negotiation interactions are necessary. For example, in a large AgentScape location, the location manager can decide to split the host manager population into two virtual groups, and alternate between these groups during negotiations. This reduces the number of interactions by half (provided that no re-negotiation is necessary). For experiments regarding the performance of the AgentScape negotiation architecture, see Section 5.6.

Influencing the number of host managers in a specific negotiation process can also be used to apply resource usage policies: A location manager can selectively include or exclude host managers in negotiations. For example, to ensure that the resources are used evenly within a location, the location manager can select host managers with a low number of active agreements, in an attempt to increase resource usage on that host.

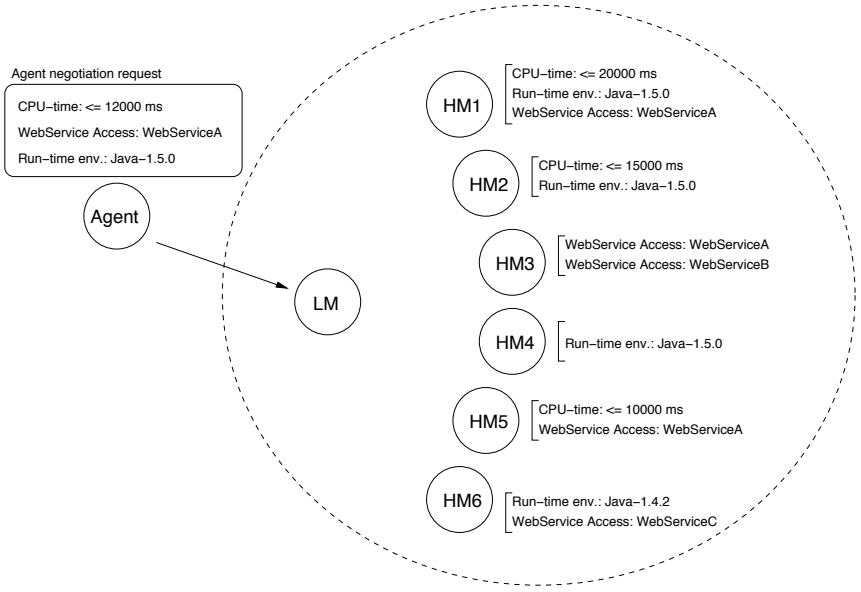

Figure 5.16: The AgentScape location, showing available resources, and the incoming negotiation request.

Example: Selecting Host Managers

In this example, an AgentScape location consists of six hosts. Each host offers a number of resources. The goal of the example is to illustrate the host selection process within the location manager, after having received a negotiation request from an agent. Figure 5.16 shows the AgentScape location, and the negotiation request of the agent. In the figure, the resources available on each host are shown. In the example the location manager has collected the advertisements from each of the host managers, and has created a combined advertisement. The combined advertisement has been used by the agent to create its negotiation request. Based on the advertisement information, the location manager selects all relevant host managers. Figure 5.17 shows these five host managers. The resources that match have been depicted.

In this example, the location manager applies a *negotiation policy*, aimed at reducing the overall negotiation time and resources required for each negotiation process. This policy is implemented in the following way: The number of selected host managers is reduced by removing the host managers with the lowest number of matching resources, until at most half of the selected host managers , thus reducing the number of negotiation requests in this example by two. Figure 5.18 shows that three hosts are selected. Finally, to each of the selected hosts, negotiation requests are communicated. Figure 5.19 shows the outgoing negotiation requests.

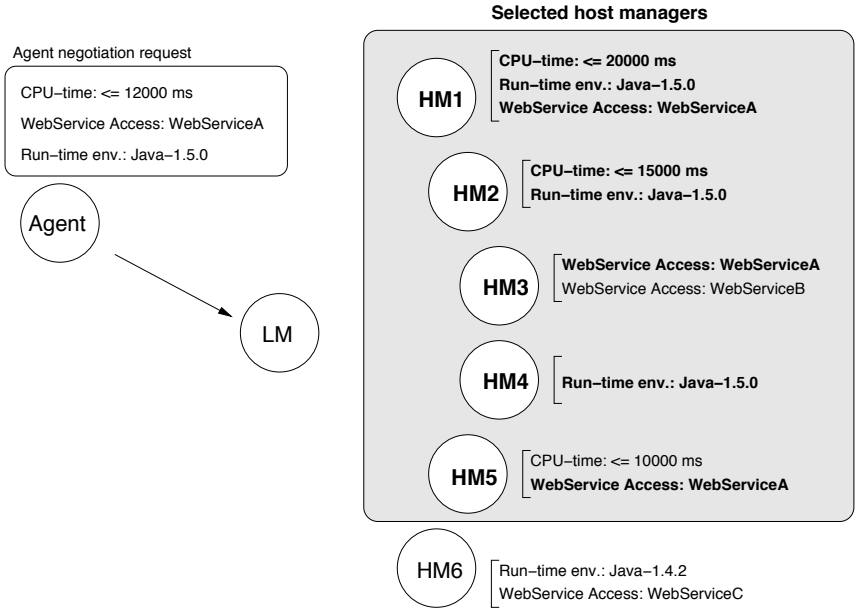

Figure 5.17: The selected host managers.

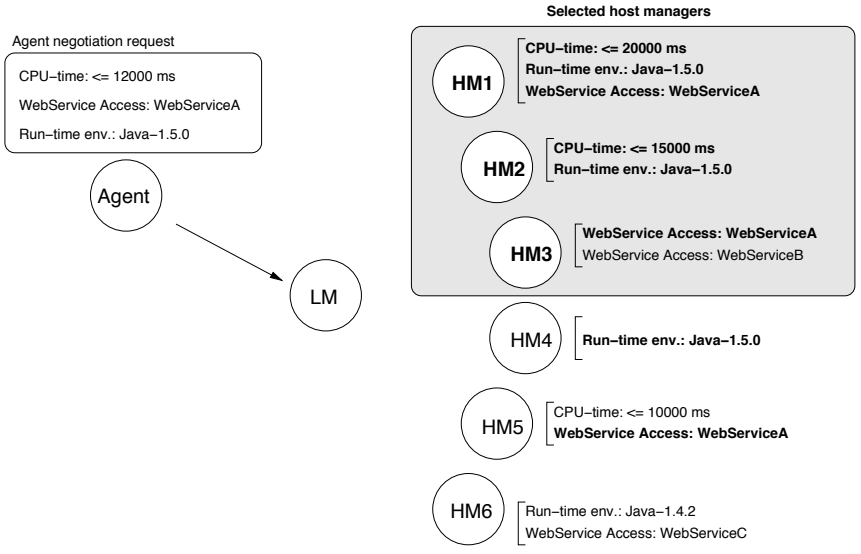

Figure 5.18: The definitive selected host managers.

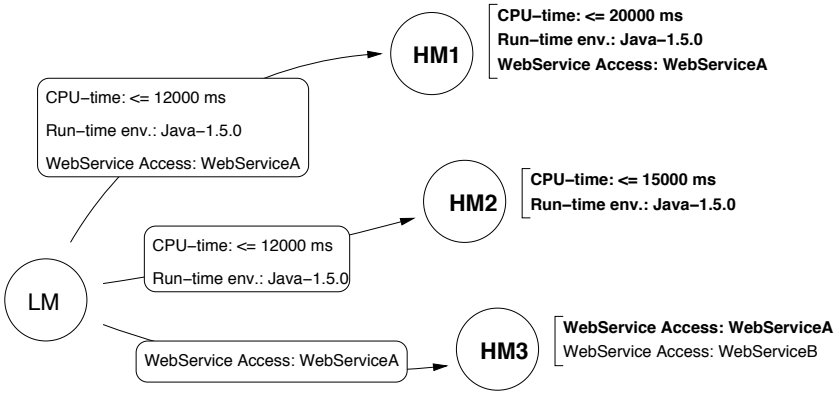

Figure 5.19: The negotiation requests sent out to the host managers.

B: Selecting and Combining Host Manager Offers

After the location manager has sent out the negotiation requests as described above, it gathers the negotiation offers from the host managers. The location manager now selects which offers it will use to create a negotiation offer for the agent. This process is governed by two types of knowledge:

1. Utility calculation mechanisms;
2. Location-wide selection policies.

1. Utility Calculation Mechanisms Upon receiving negotiation offers from the host managers, the offers are compared to determine the best offer. For example, if two host managers return a negotiation offer containing a *CPU-time* resource, the offer containing the highest CPU-time value is selected.

If negotiation offers contain multiple resources, the location manager uses knowledge about the utility of the combined resources to compare the combinations with each other, and select the combination with the highest total utility.

2. Location-wide Selection Policies Selection policies influence the offer selection process. For example, a location manager applies a 'load-balancing' policy by selecting the offer of the host manager with the lowest number of active agreements, if several host managers return a similar offer. This policy results in agents being placed at hosts with the lowest number of agents within the location.

Example: Host Offer Selection

This example continues the example used in the previous section. The example ended with negotiation requests being sent to three hosts within a location (see Figure 5.19). In response to these requests, the host managers return their negotiation offers. These offers are shown in Figure 5.20.

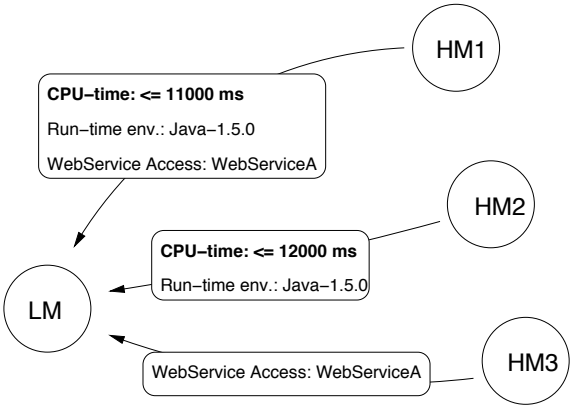

Figure 5.20: The negotiation offers received the host managers.

The location manager examines the offers: The offer from HM1 provides all requested resources, but the requested CPU-time of 12000 ms has been reduced in the offer to 11000 ms (lower than specified in HM1's advertisement). The offers from HM2 and HM3 separately do not provide the required resources, but combined they do. The location manager has two options from which to choose. Figure 5.21 shows the two offers: Offer 1 is the single offer from HM1. Offer 2 is the combined offer from HM2 and HM3.

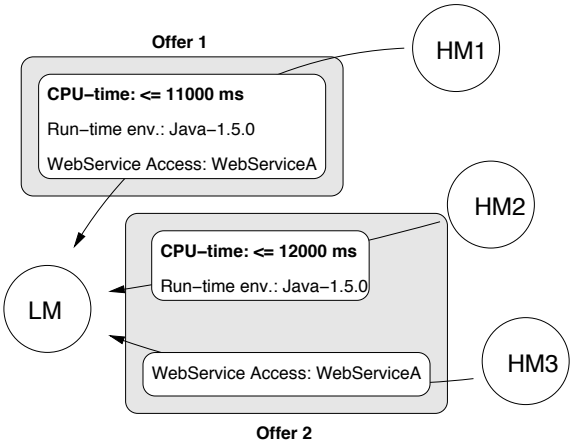

Figure 5.21: Two offer bundles.

Offer 2 is preferable over Offer 1, as Offer 1 provides a lower CPU-time value. However, in this example, the location manager applies an organizational policy (the reasons behind this decision are not relevant for this example) which specifies that when two offers are comparable, an offer from a single host manager is preferred over an offer composed of multiple host manager offers. The location manager decides in this case that the offers are sufficiently alike (it considers the 11000 ms offer similar to the 12000 ms offer), and chooses Offer 1. The offer is communicated to the agent, and HM2 and HM3 are informed that their offers have been rejected. Figure 5.22 depicts these messages.

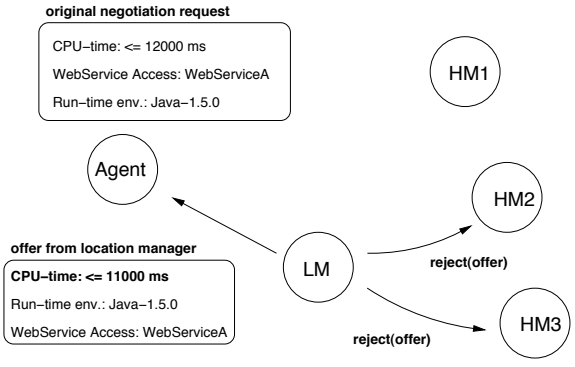

Figure 5.22: The selected offer, and rejection of the other offers.

5.4.6 Agreement Management

The location manager is responsible for maintaining information on agreement offers that have been offered to agents, and agreements that have been accepted by agents and have been implemented. Also, any incoming *agreement status requests* from agents are processed.

When an agent informs a location manager that it accepts a particular offer, the agreement management component of the location manager retrieves information on which host manager offers were used to create the negotiation offer accepted by the agent. For each of these offers, the location manager contacts the host manager, and accepts the offer. Any other offers are rejected, as described in the previous section. Figure 5.23 shows the acceptance of an offer consisting of two host manager offers.

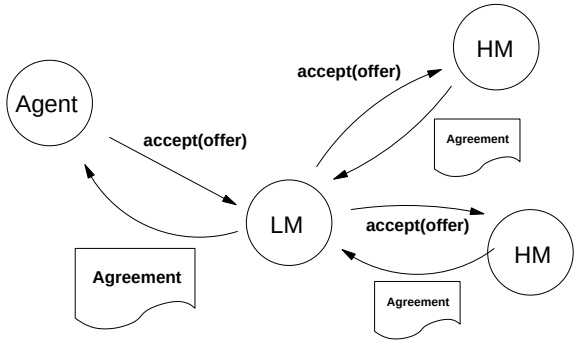

Figure 5.23: Acceptance of an agreement offer in AgentScape.

Finally, the location manager receives agreement documents from the host managers, combines them, and forwards the combined final agreement to the agent.

The location manager is also responsible for handling requests from agents concerning the status of established agreements. To determine the status of an agreement, the location manager requests the status of the underlying host manager agreements, by calling *requestAgreementStatus()* on the host managers implementing the agreements (see Figure 5.24). The resulting status information is combined and returned to the agent. The retrieval and combination of agreement status information is implemented in the AgentScape middleware in a coarse-grained form: Agreements can either be *pending*, *active*, or *violated*. Future versions of the implementation will enable detailed status information concerning the different resources specified in advertisements.

5.5 Service Provider

Host Managers are the *service provider agents* in the AgentScape negotiation architecture. The following main negotiation tasks are performed by a host manager:

1. Maintaining advertisement information.
2. Handling incoming negotiation requests from the location manager.

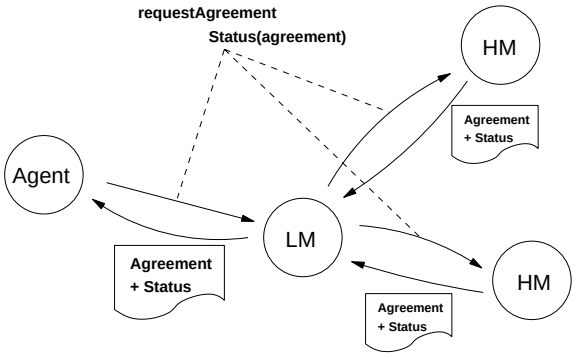

Figure 5.24: Requesting agreement status information in AgentScape.

3. Handling resource reservations that are made during the negotiation process.

4. Implementing accepted offers in the middleware (and rejecting declined offers).

These tasks are performed by the *negotiation module* (NM) and the *service management module* (SMM) of the host manager. The *host management module* (HMM) is responsible for maintaining information on the middleware (e.g. agent servers, web service gateway, etc.). This information is used by the SMM to determine the availability of resources, and by the NM to determine the content of advertisements. The *agent management module* (AMM) maintains information on the agents running on the host (e.g. lifecycle information), and handles migration of agents to and from the host. Figure 5.25 shows these modules, and the other modules of the host manager.

5.5.1 Negotiation Module

In this section, the components of the *negotiation module* of the host manager are discussed. After this, the *service management module* is discussed.

Advertisement Management

The host manager is responsible for providing an advertisement containing the descriptions of available AgentScape resources on the host, and possible restrictions imposed by administrators. The configuration of an AgentScape host is dynamic (middleware services may be added/removed or may crash). For every change in the host configuration, the host manager updates the host advertisement to reflect the change. Also, host administrators may change the usage policies of specific services (e.g. decreasing the maximum allowed CPU-time which may be used by agents).

In AgentScape, host administrators can directly access and modify the advertisement information to reflect policy changes. Currently, the advertisement information is stored in a configuration file, which is periodically checked, to determine if any changes have been made by the administrator. Future versions will let administrators define resource policies using a policy language, and translate provider advertisements on demand.

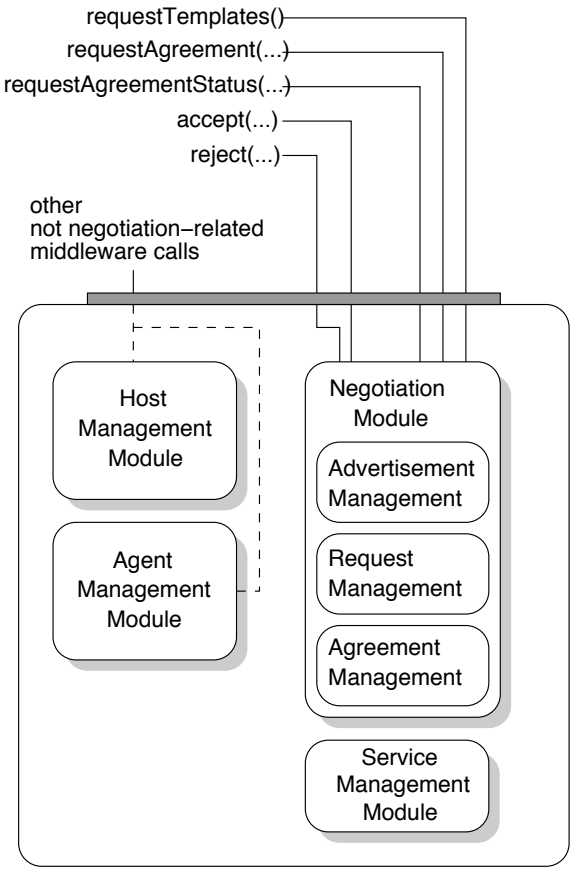

Figure 5.25: Host manager components.

Request Management

When a host manager receives a negotiation request from its location manager, the request is validated against the advertisement on which it was based. If validation succeeds, an offer is created. The contents of the offer is based on (i) the current state of the requested resources, and (ii) resource policies concerning the resources.

Information about the current state of the resource is obtained through the *service management module* described in section 5.5.2. For example, if a request for web service access is received, the state of the web service access resource is determined. If the resource is used heavily by other agents, the host manager can decide to not process the request further.

In addition to the resource state information, *resource policies* also govern the request management process. For example, a policy specifying that a maximum of 20 agents may access a resource may be defined. When this amount has been reached, no additional offers are made until some of the agents have left the host. These policies are currently specified directly within the host manager, but future versions will enable administrators

to define the policies explicitly using a policy language.

When a negotiation offer has been created, the host manager must ensure that the resources offered can be provided. For this purpose, the host manager uses the *service management module* to reserve resources. For example, when an offer is made for a Java *run-time environment*, the host manager must make sure that the agent server that implements this resource, reserves space for the agent for which the offer is intended.

Agreement Management

Agreement management is responsible for maintaining information on outstanding offers and active agreements. Each outstanding offer has a duration associated with it, specified in the policy implemented within the host manager. When this duration is exceeded, the offer is removed, and all associated resources are released by the *service management module*.

Active agreements are monitored using the *service management module* to determine if agreed resources are being delivered, or whether violations have occurred. In AgentScape, the current policy is: When part of an agreement is considered to be violated (e.g. an agent has consumed the allocated amount of CPU-time), the entire agreement is considered violated. Violations of an agreement result in termination of the agreement. When terminating an agreement, the *service management module* releases the resources associated with the agreement.

When an agreement is violated or expires, agents are not informed of this, and agents are denied access to resources until a new agreement is established. Agents are not informed as they are considered responsible for monitoring their own agreements and to take timely actions to negotiate a new agreement if required. Other policies are possible.

5.5.2 Service Management Module

The *service management module* provides the negotiation module with functionality to (i) determine the current state of resources, (ii) create resource reservations during negotiation (if required), and (iii) implement established agreements at the service level.

For these purposes, for each resource in AgentScape, a *resource handler* is created and maintained by the *service management module* (see Figure 5.26). Each resource handler supports four basic operations:

- *Object getMonitorValue(monitorObjectID)*: This generic monitoring call is used to retrieve resource specific monitoring data on the resource the handler represents. For example (in the case of a CPU-time resource handler), the call can be used to request data on the consumed CPU-time of an agent. The return value of this call is a resource handler specific data-type.
- *reservationID reserve(serviceDescriptionTerm)*: This call is used to reserve resources during negotiations. The argument of the call contains an AgentScape specific resource description term, taken from the negotiation offer for which the reservation is made, specifying the amount of the resource that is to be reserved. The call returns a *reservation identifier*, which can later be used by the negotiation module to implement or release the reservation.

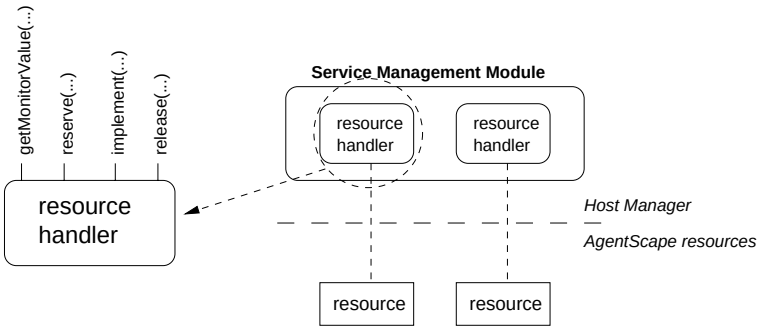

Figure 5.26: Resource handlers in the Service Management Module.

- *implementedID implement(reservationID)*: This call is used to implement a reservation made earlier by the negotiation module. After this call, the resource is ready for access by the agent. The call returns an *implemented identifier*, which can be used to release the resource.

- *release(reservationID | implementedID)*: This call is used to free any reserved or implemented resources, in case of violation or expiration of an agreement. When an identifier of an implemented agreement is supplied as the argument, an agent can no longer make use of the resource. When an identifier of a reservation is supplied as the argument, the reservation is removed.

The resource handlers currently implemented in AgentScape are discussed below.

CPU-time Resource Handler

For each agent server on a host[1], the host manager contains a CPU resource handler. The handler enables *negotiation module* to monitor CPU-time consumption, using the *getMonitorValue(...)* call: The resource handler periodically retrieves monitoring information from the agent server, detailing the amount of CPU-time that agents have consumed on the agent server. Figure 5.27 shows this process.

When a reservation is made for the CPU-time resource by the *negotiation module* using the *reserve(...)* call, the resource handler initializes a counter that keeps track of the consumed CPU-time of the agent. No active CPU scheduling is performed by the resource handler (but this could be added in future versions).

When the *negotiation module* calls the *implement(...)* call, the counter is started, and will keep track of the consumed CPU-time of the agent.

Monitoring

The CPU resource handler monitors the consumed CPU-time of agents on the agent server it is assigned to. Monitoring is performed using polling (as opposed to event-based monitoring). Polling intervals are not based on regular intervals, but instead are calculated

[1] Currently, only the Java agent server implementation allows for CPU monitoring.

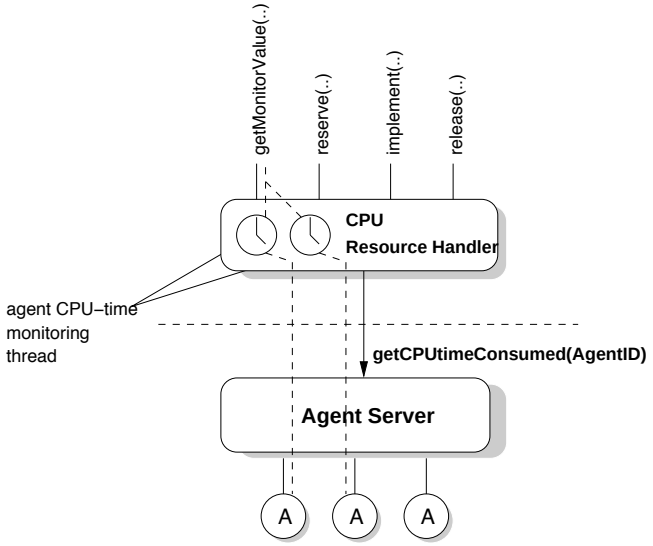

Figure 5.27: CPU resource handler.

based on the expected consumption rate of the CPU-time resource: An estimate is made of the amount of *wall clock* time the consumption of the assigned CPU-time will take. As an example, consider an agent that has agreed to use at most 2000 ms CPU-time. The initial polling interval will be based on the allocated CPU-time value: Polling of the consumed CPU-time will occur after 2000 ms (wall-clock time) have passed. If at that point in time the consumed CPU-time is less then the wall-clock time (i.e. the measured CPU-time value at that time reveals that the agent has been scheduled on the CPU for a total of only 1400 ms), a new polling interval of 600 ms is scheduled.

To enable the monitoring of agent CPU consumption, the JVM Tool Interface (JVMTI) of the Java virtual machine (see Figure 5.28) is used. The JVMTI provided by the JVM provides calls to monitor applications running in the Java virtual machine.

The JVMTI interface is however not directly accessible from within the JVM: Java applications have to use native calls (using JNI) to access the interface. For this purpose, a small C library is implemented which exposes the *GetThreadCpuTime(ThreadIdentifier)* call offered by the JVMTI. This library performs the JVMTI call to obtain the consumed CPU-time value for the agent thread.

Other methods of determining CPU consumption of Java threads are available, but either require modifying the JVM [16], or using operating system specific mechanisms (e.g. using */proc* on a linux based system).

The current implementation of the AgentScape agent server does not offer facilities for fine-grained scheduling of agents for regulating CPU-time consumption, but instead offers control of the agent life cycle, as an indirect method of limiting CPU consumption: When a violation of the agreed CPU-time is detected, the current policy of the *negotiation module* is to kill the agent, resulting in its removal from the agent server, and freeing the CPU resource. Finally, the *negotiation module* also then calls the *release(. . .)* call, which

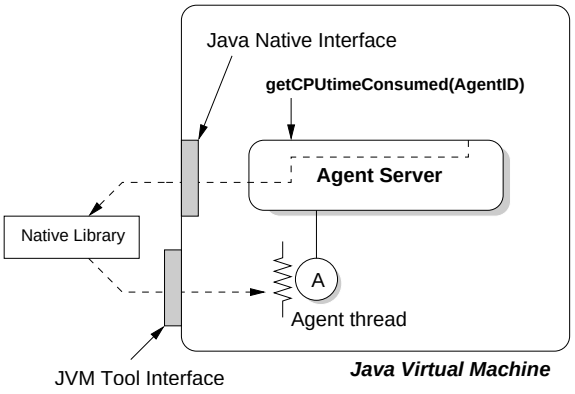

Figure 5.28: Monitoring of agent CPU consumption.

removes the counter in the service handler, as it is no longer in use. Other policies are also possible, for example giving agents a warning prior to their removal, enabling the agents to establish a new agreement, or migrate to another location.

Run-time Environment Resource Handler

Analogous to the CPU-time resource handler, the run-time resource handler is also related to agent servers. The handler maintains a counter to indicate the number of agents currently using the resource (i.e. running on the agent server). The service handler monitors the number of agents on the agent server using the *getNumberOfAgents()* call on the agent server. Figure 5.29 shows the resource handler in relation to the agent server.

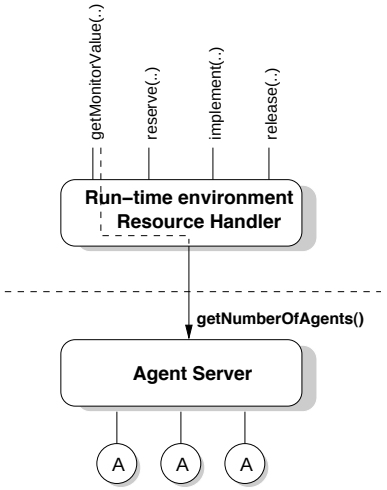

Figure 5.29: Run-time environment resource handler

The *negotiation module* uses the *getMonitorValue()* call to retrieve the current number

of agents, for determining during negotiations whether additional agents are allowed on the agent server. The *reserve(..)* call of the resource handler results in resources being reserved at the agent server, ensuring that the agent server has sufficient resources available to run the agent when it is started (e.g. memory/communication bandwidth/message buffers/etc.). The *implement(..)* call activates the reservation, and ensures that the agent can be started on the agent server. The *release(..)* call ensures that any resources associated to an agreement or reservation, are freed. In the current AgentScape implementation however, agent servers do not support advance allocation and reservation of resources for exclusive usage by agents.

Web Service Resource Handler

This resource handler enables the negotiation module to implement agreements concerning access to web services through the web service gateway. The handler monitors the current load of the gateway, using the *getCurrentLoad()* call (see Figure 5.30). This information can be requested by the negotiation module through the *getMonitorValue()* call, to determine if the load of the gateway allows additional access by agents, or not.

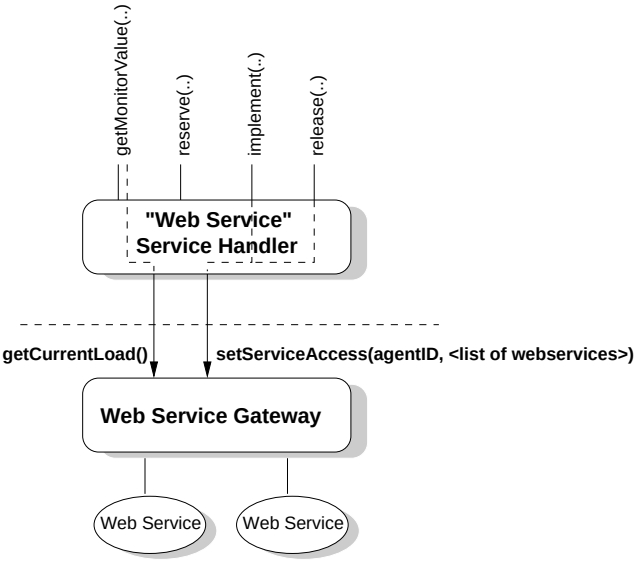

Figure 5.30: ”Web Service” resource handler

The current implementation of the web service gateway in AgentScape does not support advance allocation and reservation of resources associated to web service access (e.g. call rate, bandwidth, etc.). Currently an access control list is maintained, specifying which web services may be accessed, and which may not. Reservations are therefore not applicable, as no actual resources are consumed at the web service gateway. Therefore the *reserve(. . .)* call is not implemented for this resource. The *implement(. . .)* call results in the resource handler calling *setServiceAccess(agentID, webservice_name, true)* on the

web service gateway. This call instructs the gateway to allow the agent indicated by the first call argument access to the web service indicated in the second argument of the call. After this, the agent that has established the agreement is able to use the gateway, but only for the agreed upon web service. Any communication to other web services sent by the agent are not forwarded by the gateway. The *release(...)* call results in the resource handler calling *setServiceAccess(agentID, webservice_name, false)*, resulting in the agent not being denied access to the specified web service.

5.6 Implementation and Experiments

The goal of the experiments described in this section is to assess the performance of the AgentScape negotiation infrastructure. The experiments are divided into three categories. The first set of experiments measures the response times of the various subparts of the negotiation infrastructure. In the second set of experiments, the complete infrastructure is subjected to increasing load, and measurements are performed to compare the response times of the negotiation infrastructure in different load situations, from the perspective of the agents negotiating with architecture. The third set of experiments describes a specific implementation of a location manager negotiation policy, that balance agents across agent servers within an AgentScape location.

5.6.1 AgentScape Middleware

The current version of AgentScape provides both a C/C++ and a Java implementation of the middleware kernel process[2]. The middleware is implemented in Java. The experiments described in this section use the Java version of the kernel, together with the Java middleware layer. All experiments are performed on the DAS-2 cluster of the Vrije Universiteit Amsterdam. The cluster consists of 72 dual Pentium-III nodes connected by Fast Ethernet (the cluster also offers a high speed Myrinet network infrastructure, but this was not used in the experiments).

5.6.2 AgentScape Negotiation Experiments

To determine the performance and potential performance bottlenecks of the negotiation infrastructure, a number of experiments are performed to measure the response times of individual components within the two-tiered negotiation infrastructure, as well as the response time of the complete negotiation infrastructure. In this section, first, the response times of the low-level communication infrastructure provided by the AgentScape kernel are determined, upon which the negotiation infrastructure is built. After this, response times of the host manager negotiation process are examined. Finally, response times of the complete negotiation infrastructure are measured.

[2]For a performance evaluation of the Java and C/C++ kernel implementations, see [91]

Byte array size (bytes)	0	3000	6000
Response time (ms)	3	3	3

Table 5.14: *Mode* of the response times (ms) using the kernel test calls.

5.6.3 AgentScape Kernel Experiments

To determine the performance of the AgentScape middleware infrastructure underlying the negotiation components (i.e. the kernel), experiments are performed in which two dummy middleware processes are instantiated on two AgentScape kernels running on separate hosts. The aim of the experiment is to determine the time required for a single communication call from one middleware service to another. A *ping()* call which returns an array of bytes is provided by one the dummy services. Figure 5.31 shows the configuration used in the experiment.

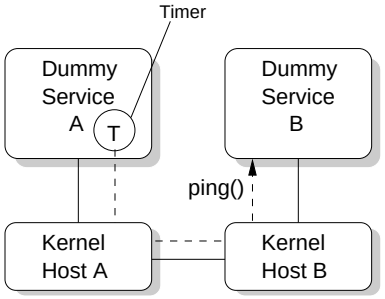

Figure 5.31: 'Ping' call setup.

A number of experiments compare response times, for varying lengths of the byte array returned by the call: 0, 3000, and 6000 bytes. These values are the size of the negotiation documents that are transferred through the kernel during the negotiations (approximately 3000 bytes). Each experiment consists of 5000 calls of the *ping()* method performed sequentially. Experiments are repeated five times. Response times are measured using a millisecond resolution.

Figure 5.32 shows the results of an experiment, for a returned byte array length of 3000 bytes. Figure 5.33 shows the distribution of the response times in the same experiment. These results show that the distribution of the response times contains a small number of relatively high-value 'outliers'.

Due to these outliers, using the average of these response times would give an atypical performance measure. The *mode* however gives the performances measure most likely to be observed by the AgentScape negotiation framework. Table 5.14 shows the *mode* of the response times obtained in the above experiments.

The results show that the time for communicating from one middleware process to another takes approximately three milliseconds. This only accounts for communication: No pre- or post-processing of the document is taken into account (this is discussed in the upcoming experiments). For more experimental results concerning the performance of the AgentScape kernel communication infrastructure, and other kernel functions, see [91].

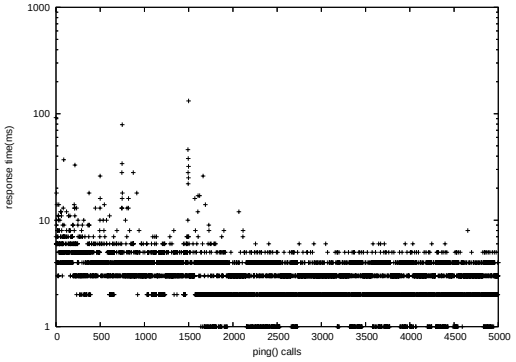

Figure 5.32: Example of a 'ping' call experiment (3000 bytes).

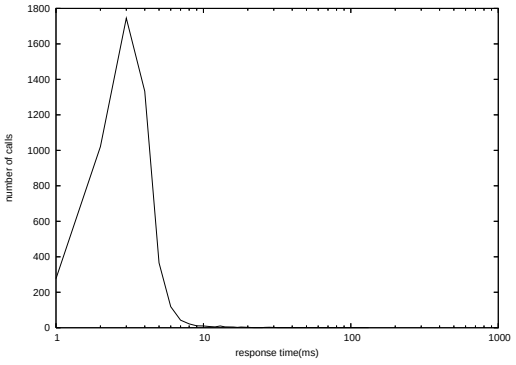

Figure 5.33: Distribution of response times shown in Figure 5.32.

5.6.4 Host Manager Negotiation Experiments

In this section, the performance of the host manager is examined, by measuring the response time of typical negotiation calls to the host manager. In the experiments, a host manager and a location manager are instantiated on two kernels running on separate hosts. The location manager performs negotiation calls on the host manager, and measures response times. Three sets of experiments have been performed:

1. Empty negotiation requests: The location manager performs *empty* negotiation requests, in which a negotiation request is sent to the host manager with no service requests specified. As a result, the host manager only performs basic request verification and handling, and does not apply any policies. An empty agreement offer is returned by the host manager. These experiments are aimed at determining the

Experiment set	Response time (ms)
1. Empty request	6
2. CPU-time request	19
3. CPU-time + accept	27

Table 5.15: *Mode* of the response times for the negotiation calls.

minimal time required by the host manager to process an agreement request.

2. CPU-time negotiation requests: In these experiments, the location manager performs negotiation requests, which contains CPU-time resource requests. The host manager processes the request and applies a straightforward CPU-time policy: The requests are accepted unconditionally. These experiments are aimed at determining the time required for handling a typical negotiation request, and returning an answer.

3. CPU-time negotiation and acceptance: The consumer performs the same negotiation request as in the previous experiment, after which the location manager performs an *accept(...)* call to the host manager, accepting the obtained offer. These experiments aim to measure performance of the host manager under normal negotiation situations, in which agreements are requested and accepted.

Figure 5.34 shows an overview of the calls performed in the different experiments.

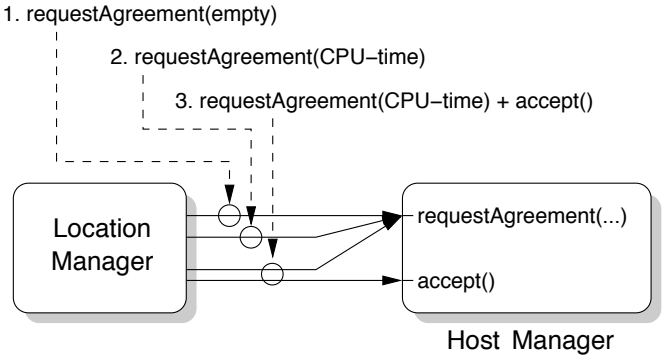

Figure 5.34: Measuring response times between a location manager and a single host manager.

As in the kernel experiments, each experiment consists of the location manager issuing 5000 sequential requests to the host manager, and measuring the individual response times. As shown in Figure 5.34, in the last set of experiments, each request consists of 2 calls (request and accept). In this last set of experiments the total response times over both calls is measured. Experiments have been repeated 5 times. Table 5.15 shows the results.

From these results, the overhead generated by the host manager can be determined: The kernel level experiments discussed earlier indicate that a single call between middle-

ware services takes approximately 3 milliseconds. For the experiments described above this means that:

- Experiment set 1 (empty request): In this case, the overhead generated by the host manager due to the processing of the empty agreement, and returning an empty agreement, is 3 milliseconds: 6 ms minus 1 middleware communication (3 ms).
- Experiment set 2 (CPU request): In this case, the overhead generated by the host manager due to processing a request for the CPU-time resource, is 16 ms.
- Experiment set 3 (CPU request + accept): In this case, two communication calls are performed, which result in approximately 6 milliseconds of communication time. The overhead generated by the host manager (creating a negotiation offer, and subsequently implementing it) in this case is 21 ms.

The results are an indication of the overhead generated by the host manager middleware service in the negotiation process. It is clear that host manager overhead affects negotiation response times. The amount of time a host manager spends on determining a negotiation offer depends highly on the complexity of the negotiation policies. In the above experiments, a straightforward policy is used, in which the requested CPU-time is blindly accepted by the host manager. In situations in which a host manager uses more complex resource scheduling algorithms, response times will increase.

5.6.5 Full Negotiation Architecture Experiments

In the following experiments the complete negotiation infrastructure is tested in a realistic setting. Experiments are performed with different AgentScape configurations consisting of multiple host managers, and with varying consumer load, by varying the number of agents requesting agreements. For each experiment, an AgentScape location is deployed across the nodes in a cluster: One node is configured to run the AgentScape location manager, the other nodes are configured to run either host manager processes or consumer processes (i.e. agents). Consumers request access to the CPU-time resource, and subsequently accept the offer which is returned by the location manager.

In these experiments, all incoming agent negotiation requests are sent to all host managers within the location (no 'host selection' is performed by the location manager). Subsequently, the location manager selects one host manager with whom to negotiate, and forwards the offer returned by this host manager to the requesting agent. The offers from the other host managers are not further processed: This results in expiration of these offers when their duration has been exceeded. Response time measurements are performed for location configurations with varying numbers of host managers and agents, to determine the increase in response times when negotiation load increases. Agents run concurrently, each on a single host, and each perform their negotiation calls sequentially. During the experiments, for each call pair (request and accept), the total response time is measured in milliseconds by the agents. Figure 5.35 depicts the setup of the experiments.

To subject the negotiation infrastructure to increasing load, experiments are performed with an increasing number of host managers and agents, to a maximum of 32 host managers and 32 agents. Each agent is configured to perform 1000 sequential negotiation

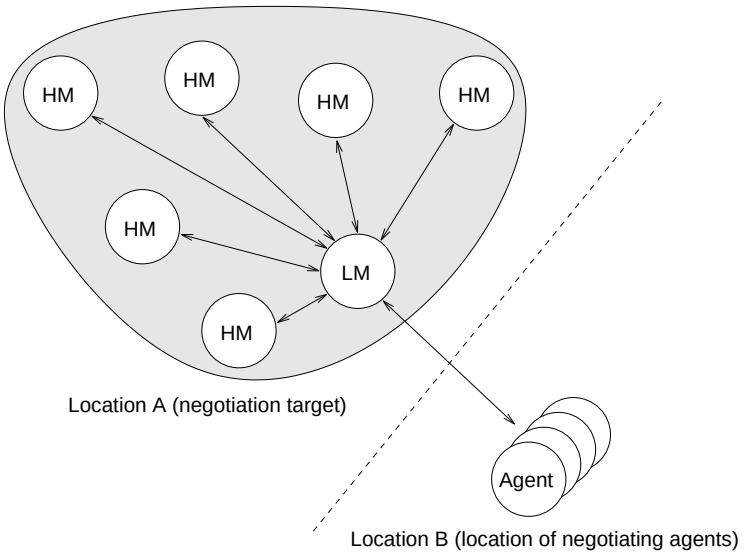

Figure 5.35: Experimental setup.

#Agents - #Host Managers	*1*	*2*	*4*	*8*	*16*	*24*	*32*
1	40	51	54	61	81	101	139
2	49	53	60	77	126	180	234
4	62	64	83	117	226	365	437
8	92	96	137	207	452	675	860
16	89	178	266	450	859	1412	2043
24	91	285	415	719	1834	2564	3539
32	101	276	562	894	2040	3443	4799

Table 5.16: *mode* of the average negotiation response times (ms).

requests. Experiments are repeated 5 times. As in the previous experiments, the *mode* of the response times is taken as representative of the response time distribution, to remove excess influence of any outlier values due to external conditions. Table 5.16 shows the results.

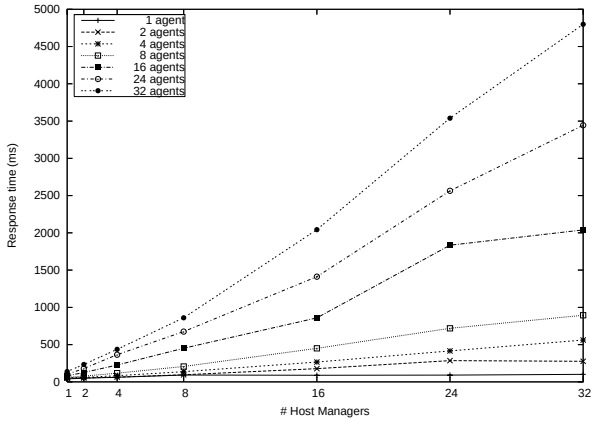

Figure 5.36: Increasing host managers.

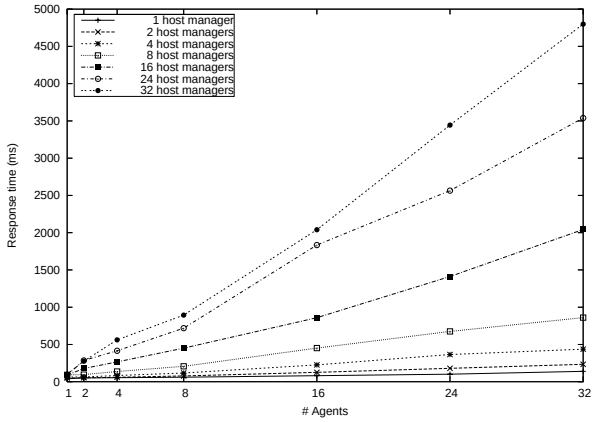

Figure 5.37: Increasing agents.

To acquire a better understanding of the impact of increasing the number of host manager versus increasing the number of agents, two plots are made: Figure 5.36 shows the response times when the number of host managers is increased: Each line represents a fixed number of agents (1, 2, 4, 8, 16, 32). Figure 5.37 shows the response times when the number of agents is increased, while keeping the number of host managers fixed (1, 2, 4, 8, 16, 32). The figures indicate that increasing the number of agents leads to an approximately linear increase in response times. Similarly, increasing the number of host managers also leads to an approximately linear increase in response times. Figure 5.38

shows the increase in response times when simultaneously increasing the number of host managers and agents, up to 40 host managers and 40 consumers.

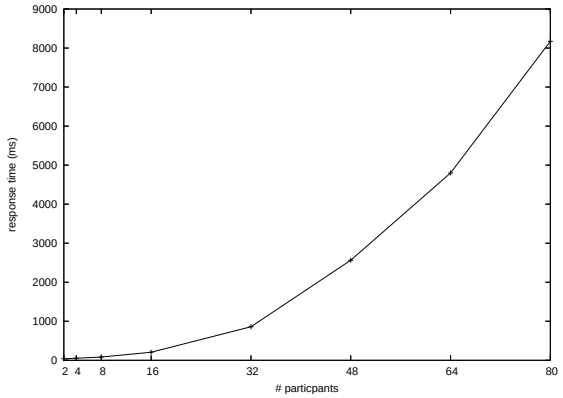

Figure 5.38: Increasing both agent and host manager populations.

The results show that increasing the number of participants (agents + host managers) leads to a non-linear increase of response times in the negotiation process. The increase in the number of negotiation interactions that occur in the experiments is an important factor. The results are discussed further in the next section.

Performance Experiments Discussion

In the first set of experiments, the increase in negotiation response times is measured under increasing negotiation 'load'. Increase of the load of the AgentScape negotiation infrastructure is achieved in two ways: By increasing the number of agents negotiating through the infrastructure, and by increasing the number of host managers in the target location. The results show that when increasing both the number of agents and host managers, response times increase non-linearly. The main cause is the increase in negotiation communication interactions:
The number of negotiation interactions in the experiments can be determined as follows: Each negotiation interaction of an agent with the location manager in the experiment leads to two negotiation calls between the agent and the location manager (request and subsequently accept/reject). In addition, the location manager sends out requests to all host managers within the location, for each request from an agent. Finally, an *accept* call is performed to one host manager, the host manager selected to provide the requested CPU-time resource. As mentioned earlier, no explicit rejection calls are made to the other host managers in these experiments. The number of negotiation calls resulting from a configuration consisting of A agents and H host managers can be determined as follows:

$$2 \cdot A + (A \cdot H) + 1$$

To reduce the non-linear increase in response times, the number of negotiation interactions needs to be restricted. This can be achieved by implementing negotiation policies at

the mediator that reduce the number of host managers involved in negotiations, effectively reducing the total number of communication interactions between the mediator and host managers within the location. For this purpose, a location manager can use the information available in the advertisements obtained from the host managers within the location, to intelligently select with which host managers to negotiate.

5.6.6 Agent Load Balancing Experiments

This section describes experiments in which a location manager and host managers implement negotiation policies that influence the allocation of resources to agents, with the aim of distributing agents across the available resources within a location. The first set of experiments centers on the ability of the negotiation architecture to accommodate domain-wide resource policies. The second set of experiments focuses on the use of the negotiation architecture to apply "quality of service" policies using individualized host policies.

Experimental Setup

A distributed AgentScape location is set up with nine hosts. Eight hosts are configured to run both a host manager and an agent server, and one host is configured to run a location manager. The location manager implements the domain coordinator negotiation functionality. In each of the experiments, agents migrate to the location after an agreement has been negotiated with the location manager. The hosts used for the AgentScape location are part of the DAS-2 cluster at the Vrije Universiteit Amsterdam. The agents are inserted from a host outside the DAS-2 cluster, connected by Fast Ethernet.

In the experiments, CPU-time is the subject of negotiation. In each experiment, 1000 agents are inserted into the location. For each agent, a 'desired' CPU-time amount is generated according to the Weibull distribution (scale = 3.0, shape = 2.0, mean = 26.59 seconds). The Weibull distribution is chosen as it provides a good model for 'lifetimes' of objects, in this case the agents. This value is then used to create a negotiation request which is then sent to the location. The intervals between requests of individual agents are distributed according to the Poisson distribution (mean = 2 seconds). The Poisson distribution was chosen as it provides a good model for independent arrivals of the negotiation requests at the location. Each negotiation request received by the location manager is translated into negotiation requests to the 8 host managers within the location. Each host manager then responds with a negotiation offer if the requested value is in line with the local CPU-time policy, or responds with an empty offer if the requested value is not in line with the policy. In the experiments, the load on a host is represented as the number of agents running on a host, measured at one second intervals.

Experiment 1: Domain-wide Negotiation Policy

Domain policies facilitate the distribution of computational load across available hosts in the environment. Two straightforward types of policies are based on the principles of: (1) time-division, in which computational load is scheduled for execution at different times, and (2) space-division, in which computational load is scheduled on different hosts. In these experiments, a round-robin (space-division) negotiation policy is applied: The aim

of this policy is to distribute all agents evenly throughout the location. In the experiments, a location manager collects negotiation offers made by the hosts, and applies a round-robin load balancing policy to select one of the offers made by the host managers. This offer is then sent back as an answer to the original negotiation request of an agent. After acceptance of the agreement, the agent migrates to the host that has been selected during negotiation. The agent then starts to consume CPU-time by performing some predefined CPU-bound computations. When the CPU time delegated to the agent in the agreement has been consumed, the agent is stopped and removed from the host. In this experiment, hosts are configured with a negotiation policy dictating that all negotiation requests from agents should be accepted, regardless of the requested CPU-time value.

As a measure for the balance of the load within the AgentScape location, the "Load Balance Metric" is used, as described by Bunt and Eager [20]. This metric is calculated by taking the weighted average of peak-to-mean server load ratios. This ensures that a larger imbalance during high-load situations has a greater effect on the LBM measure than a smaller imbalance during lower-load conditions. The value of the LBM measure ranges from the number of servers (8 hosts in the experiments) to 1, where a lower value represents a higher balance. In Fig. 5.39, the LBM values are graphed, calculated over 10 second intervals. The figure shows that a consistent balance is achieved within the location using the round-robin policy, during the insertion of agents as described in the experimental setup. Near the end of the experiment, load balance can no longer be enforced, as all agents have been inserted. This is shown in the graph by the sharp increase of the LBM value.

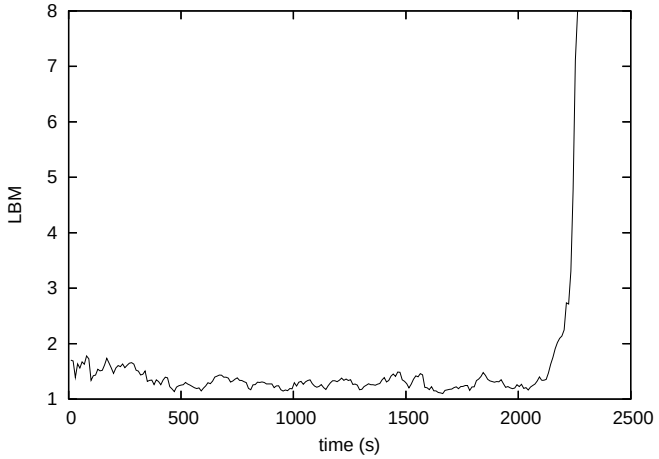

Figure 5.39: LBM over 10 second intervals using round-robin negotiation policy.

Experiment 2: Differentiated Host Policies

In the second set of experiments, negotiation policies are applied to implement a quality of service policy aimed at improving responsiveness for agents with a relatively short lifetime (below the mean value as described above). In the experiments, two different host policies are used, resulting in two different host groups: a policy allowing only requests

# below mean hosts	avg. for below mean agents %	avg. for above mean agents %
2	8.3	38.6
3	24.9	13.2
4	76.3	9.7
5	87.7	5.8
6	90.9	4.5
reference	14.5	16.2

Table 5.17: Quality of service percentage results of the CPU-time differentiated host policy experiments.

below the mean CPU-time value (short jobs), and a policy allowing only requests above the mean CPU-time value (long jobs). In each experiment, the number of hosts in the groups is varied. Agent 'desired' CPU-time generation and the intervals between agent requests are kept the same as in the previous set of experiments. Given the requested CPU-time value, the appropriate provider group is selected. Within the group, the round-robin selection policy as described in the previous experiments is used. Table 5.17 shows the results of these experiments. The first column depicts the number of hosts accepting only agents with a CPU-time value below the mean. The second and third column present a quality of service percentage for agents with a below-mean and above-mean CPU-time value respectively. The quality of service percentage metric is defined as the actual CPU-time agents have consumed divided by the "wall clock" time agents have spent on a host. The results in the table show the mean over three experiments. A high quality of service percentage of 100 % indicates a perfect quality of service where the resource is completely available to the agent (the agents in the experiments are CPU bound, and, e.g., not waiting for I/O or network communication). A low quality of service percentage means that the agent has to compete with other agents (or generally tasks) to access the resources.

The values in the bottom row are obtained from the load balancing experiments presented in the previous section, in which no differentiation was made based on CPU-time values, and agents could be placed on all hosts. This can be seen as a "reference" value, indicating the responsiveness in the undifferentiated case. From the results it can be argued that a configuration with 8 hosts, where 3 hosts accepting only agents with below-mean CPU-time values (and consequently 5 hosts accepting only above-mean CPU-time), gives agents with a shorter running time a better responsiveness, at a not too great expense for the longer running agents. For 4 hosts reserved for short running agents, the responsiveness dramatically improves with about a factor of 5 compared to the reference results, while the long running agents experience an increased turnaround time of a factor of 1.7.

Load Balancing Experiments Discussion

In the above experiments, negotiation policies are implemented and the effect of the policies is evaluated: The first experiment implements a negotiation policy at the location manager level, aimed at balancing agents across hosts in a location. This is achieved by selecting host managers to negotiate with on behalf of agents, using a round-robin strat-

egy. The second experiment implements a negotiation policy on the host manager level, aimed at increasing responsiveness for agents with short lifetimes. This is achieved by implementing policies in each of the host managers in the location, that prescribe whether to accept negotiation requests for low (i.e. below mean) CPU-time values, or high (i.e. above mean) CPU-time values.

The experiments show that negotiation policies at both the location manager level and the host manager level can be used to influence the placement of agents on hosts within a location, to achieve higher level management goals, such as load balancing and differentiation between different agent classes.

5.7 Discussion

Agent platforms in large-scale, open environments require facilities to manage resource access: Agents need to obtain assurances that the resources they require are available, and that they are allowed to use them. Resource owners and system administrators on the other hand need to be able to enforce resource access and usage policies.

The negotiation infrastructure presented in this chapter offers agents the necessary facilities to negotiate for resource access with remote locations, prior to migration. System administrators of locations define negotiation policies that influence the allocation of resources during the negotiation process. The infrastructure is implemented in the AgentScape middleware, allowing agents in AgentScape to negotiate for basic platform resources.

Negotiations are based on advertisements retrieved by agents from self-selected locations. The advertisements indicate which resources are available on remote platforms, and the negotiation space within which agents may negotiate for these resources.

Successful negotiations result in *agreement documents*, that contain detailed descriptions of the resources that can be accessed on the basis of the agreement. The agreement documents serve as proof of a successful negotiation, and are of use to both parties in the negotiation process: Agents can examine an agreement to determine what they can expect from the middleware in terms of resources. The middleware can use the document to implement the actual provisioning of the resources described in the agreement. The agreements can also serve as a legal basis, to be used to resolve conflicts between parties concerning the use of resources: Actual delivered resources can be compared to the agreed upon resources in the document, to determine whether one of the parties is accountable for violating the agreement. Based on this, subsequent steps can be taken to resolve the conflict (e.g. update trust models, award penalties, etc.). In addition to forming a legal basis, agreement documents may also be used for other higher level management goals such as accounting or monitoring.

In the infrastructure presented in this chapter, negotiations are subjected to negotiation policies that govern the negotiation process at both the organizational (i.e. AgentScape location) and local (i.e. individual host) level. These negotiation policies are aimed at regulating negotiations to ensure that resources are allocated to agents according to the resource access policies specified by both location administrators and administrators of individual hosts within the locations. In situations where many agents negotiate for

resources simultaneously and locations consists of many hosts, and response times of negotiations is an important factor, negotiation policies at the organization (i.e. location manager) level need to ensure that the number of negotiation interactions is restricted when possible, to prevent unnecessary time- and resource consuming negotiation interactions.

5.7.1 Extending the Implementation

The implementation of the negotiation infrastructure can be extended in a number of ways: Firstly, more middleware resources can be made available for negotiation, such as communication bandwidth, temporary disk space, and other resources for which it is useful to obtain guarantees.

A second way in which the AgentScape negotiation infrastructure can be extended, is by enabling agents to not only act as resource consumers, but also as providers. This allows agents to expose resources to other agents, and use the negotiation infrastructure to establish agreements with agents requiring these resources. For example, if an agent possesses knowledge/data that it would like to offer to other agents, the agent could register this as a resource at the negotiation infrastructure (i.e. providing the negotiation infrastructure with a resource handler). The AgentScape negotiation infrastructure would require a number of changes to support this: Dynamic addition/removal of resource descriptions and resource handlers has to be supported. Ideally, each agent (application) provides its own resource descriptions and handler, for the services it provides and about which it can negotiate. Agents need to be able to publish these resource handlers. The AgentScape agent interface would need to be extended with additional calls to enable this.

5.7.2 Performance and Security

An important aspect of the AgentScape negotiation infrastructure is response time. For agents, it is important that the negotiation process does not impede them performing their tasks. A tradeoff exists between the amount of time it takes for agents to establish agreements, and the benefits the negotiation process provides by enabling agent to establish resource access guarantees. Although these two factors cannot be compared easily, it is clear that the negotiation performance should be robust: Response time of the negotiation infrastructure should degrade gracefully as the interaction load increases, and provide agents with a reasonable interaction time. In AgentScape, agents are not assumed to be very short-lived processes, and resource interactions for which agreements are established are assumed to last for longer periods of time. The mediator of the negotiation infrastructure is the component which is critical for negotiation performance, as it plays a central role in negotiations within an AgentScape location. A possible approach to increase the performance of the infrastructure, is to replicate the mediator agent of the negotiation infrastructure. Replication would distribute the negotiation workload over multiple mediator instances, enabling the infrastructure to better cope with an increase in negotiation load. Replication of the mediator agent is further discussed in Section 5.7.3.

In addition to the mediator, other elements involved in the negotiation process can also affect performance: It is important to provide efficient implementations of *service han-*

dlers, as these components are used in the negotiation sequences. In these components, a tradeoff needs to be managed between complexity and negotiation efficiency.

The main security risk of the negotiation infrastructure is that agents may attempt to use agreements originally intended for other agents (e.g. to use resources the other agent has exclusive access to). A location manager stores established agreements locally to prevent this from happening: When agents request to be migrated to a location, the agreement identifier provided by the agent in the migration request is used by the location manager to retrieve the locally stored copy of the agreement. The agent identifier contained in the agreement is then compared to the identifier of the agent requesting migration. It is assumed that agents in the AgentScape middleware are uniquely identifiable, and cannot take on the identity of other agents. A drawback of this approach is that agents currently cannot negotiate agreements on behalf of other agents (which could be useful in agent applications), as the agreements are bound to the requesting agent. A possible solution for this would be to no longer bind agreements to the requesting agent, and use external authentication mechanisms and cryptographic signing of agreement documents, to enable the negotiation infrastructure to determine whether an agreement is claimed by the rightful agent.

5.7.3 Fault-tolerance

As the negotiation infrastructure is distributed over the various hosts within a location, not all failures lead to unavailability of the entire system: Failures of negotiation components of host managers lead to: (i) Active agreements concerning resources on the host are no longer available: The guarantees established in the agreements can no longer be monitored and enforced; (ii) new agreements concerning resources on the host can no longer be established. Immediately after the failure of a host manager, the location manager may still attempt to negotiate for resources with the failed host, as the advertisement information it maintains may not reflect the new situation.

Failure of a location manager leads to agents being unable to negotiate for resources available at that location. Failure however does not necessarily lead to failure of established agreements, as agreements are enforced and monitored locally by the hosts within the location. The agents may not be able to request agreement status information however, as long as the location manager is unavailable.

An approach often used to protect against failure of distributed system components is replication. Within AgentScape, replication can be applied to ensure the continuity of the location manager middleware service in the event of a failure. Overeinder et al. [73] discuss the integration in AgentScape of the DARX [64] framework, a framework for providing fault-tolerance in large scale agent systems. Currently however, AgentScape does not yet provide facilities for fault-tolerance. Replication strategies in distributed systems range from *active* to *passive* approaches: *Active* replication ensures that replicas receive all information that the original process also receives; *Passive* replication ensures that the state of replicas is periodically updated to reflect the state of the original process. Active replication allows for faster recovery, as replicas are continuously updated. Passive replication can result in loosing some state information of the original process, which was not yet transmitted to the replicas. However, an advantage of passive replication is that less

processing and communication overhead is needed for updating the state of the replicas.

For AgentScape, a semi-active replication approach is appropriate. In this approach, location manager replicas are not actively involved in the actual negotiation process (receiving all messages from host managers and agents). Instead, the 'lead' location manager communicates all established agreements and host manager template information to its replicas. In the event of a failure, the failed location manager is replaced by a replica. Only the currently ongoing negotiations are lost.

5.8 Summary

This chapter presents the application of the negotiation framework in the AgentScape agent middleware. The roles defined in the negotiation framework are mapped to the components of the AgentScape middleware: Agents are consumer agents, location managers are mediator agents, and host managers are service provider agents. In AgentScape, agents negotiate for access to low-level middleware resources at a particular AgentScape location, prior to migration: *CPU-time*, *run-time environment*, and *web service access* resources are currently supported.

The negotiation subtasks of each of the negotiation parties are described. Examples are presented for the location manager's main negotiation subtasks: Combining advertisements containing AgentScape resources, and selection of host managers in the negotiation process. Host managers provide the resources which are the topics of the negotiation process, and implement *resource handlers* which support (i) reservation of resources during negotiations, and (ii) enforcement of established agreements. Resource handlers for the three supported resources are described.

Experiments are performed to analyze the responsiveness of the negotiation architecture under increasing load: The effects of increasing the number of agents, as well increasing the number of hosts in an AgentScape location are examined. Additional 'load-balancing' experiments are performed to demonstrate the application of negotiation policies at the organizational level (through the location manager) as well as are the individual provider level (through the host managers). The experiments demonstrate the importance of regulating the number of negotiation interactions in situations where many agents negotiate with large AgentScape locations. Furthermore, the experiments demonstrate that negotiation policies can be applied to successfully achieve higher level management goals such as load-balancing and differentiated placement of agents.

5.9 Conclusions

This chapter discusses how the negotiation framework can be used to enable agents and platform administrators to reduce uncertainties with respect to access to, and use of resources: Agents negotiate resource access, prior to migration. Administrators define negotiation restrictions and negotiation policies, that are applied during the negotiation process.

The AgentScape middleware in which the negotiation framework has been implemented is an example of a next generation agent infrastructure. One of its next-generation

features is that AgentScape explicitly incorporates the *location* concept, to accommodate for the distribution of the infrastructure across multiple independent administrative domains. In AgentScape, a *location manager* is responsible for managing the hosts within a location, as well as managing agent migration to and from the hosts within the location. In this respect, the location manager is a good candidate for implementing the mediator role within the negotiation framework: Between the hosts within the location offering resources, and the agents outside the location wishing to access these resources. As a mediator, the location manager combines the resources available on the individual hosts within the location, and offers them to agents. Agents negotiate with a single mediator for access to these resources, instead of negotiating with multiple individual hosts within a location. The location manager must ensure that negotiation interactions are regulated, to prevent negotiations from consuming unnecessary time and resources. Furthermore, the location manager, as the representative of the hosts within the location, ensures that resources usage is regulated according to the negotiation policies specified by the location administrator. The 'load-balancing' experiments described in this thesis demonstrate this.

From a performance perspective, the implementation of the negotiation framework in AgentScape suffers from the performance bottleneck introduced by the location manager in the negotiation process. In situations where AgentScape locations will consist of many hosts, or where many agents negotiate for access to the location simultaneously, negotiation response times will increase considerably. Future extensions of the implementation of the negotiation framework in AgentScape will need to implement negotiation policies within the location manager that are aimed at minimizing the number of host managers involved in negotiations, while maintaining good negotiation solutions for the agents. The host manager advertisements will need to contain sufficient resource information for the location manager to base these selection policies on. Other approaches for improving negotiation response times include replicating the location manager to reduce the workload for each location manager instance, optimizing negotiation policies to reduce policy overhead, and creating advance agreements which can be used to respond more quickly to negotiation requests.

From a modeling perspective, the negotiation framework is successfully applied in the AgentScape middleware. As AgentScape already incorporates organizational concepts, the model is mapped straightforwardly on the existing AgentScape management infrastructure (location managers and host managers). Advertisements and agreements encapsulate the different resources offered by the middleware, and allow agents to negotiate for sets of resources offered by multiple hosts within an AgentScape, grouped within locations. This coincides with the world-view of agents within AgentScape: Agents do not distinguish individual hosts, and consider locations to be the smallest organizational concept within the AgentScape environment. The AgentScape negotiation infrastructure enables location administrators to regulate access to their locations by specifying negotiation policies that prevent unauthorized agents from obtaining resources access, and limit resource usage for the agents that are allowed to enter their locations. In this respect, the negotiation infrastructure can be used as a basis for facilitating higher-level AgentScape management tasks, such as enforcing security policies, accounting, and performance management.

Chapter 6

Conclusions and Future Work

Information system infrastructures will increasingly consist of large numbers of heterogeneous and distributed components, managed independently from each other and spread across multiple organizational domains. In such infrastructures, it is necessary to provide management facilities for applications to locate and acquire access to the services and resources offered by the different parties and organizations. Most of the existing management solutions are not tailored to meet the requirements that are specific for these distributed and dynamic environments.

Agent-based systems can contribute in this area, as agents capture a number of important properties relevant to this domain, such as the distributed and dynamic nature of the environment, and the autonomy of the stakeholders (e.g. applications, system administrators, organizations) in fulfilling their requirements.

This thesis presents an agent-based negotiation framework for consumers and providers of services to negotiate service access conditions. Consumers and providers negotiate through mediators. Mediators represent virtual organizations of providers. Negotiation parties do not have to know/locate each other before, during, and after negotiations. This dual mediator/organization-representative role has the advantage that organizational policies concerning the use of services can be, and are enforced by the mediator. Negotiations begin with advertisements, based on announcements of interests and result in agreements between consumers and providers of services, describing in detail the service access conditions that have been negotiated.

The negotiation framework is based on the WS-Agreement emerging Grid standard, which provides a protocol and language for negotiating agreements between individual consumers and providers of services. The framework uses WS-Agreement as a basis for its negotiation protocol and language. This enables the negotiation framework to be easily integrated with external service based infrastructures, offering similar WS-Agreement based functionality. In this thesis, the basic WS-Agreement negotiation protocol is extended to allow for more elaborate negotiations between parties, and confirmation of agreements. Furthermore, mechanisms are presented for mediation operations during negotiation, such as combining advertisements and negotiation offers.

Another important feature of the negotiation framework is the fact that a single in-

teraction protocol is used at both levels in the negotiation process. A potential drawback of this approach is performance, as was shown in Chapter 5: Using the protocol at the service provider agent level may lead to response times that are not suitable for applications in domains where timely response is an important issue. Replacing the negotiation protocol at the service provider agent level with a more restricted protocol could reduce response times, at the expense of reduced negotiation functionality and service provider agent autonomy.

An important advantage of using the same interaction protocol at the two levels of the framework is that both consumer agents, and service provider agents interact using this protocol. Both parties are not explicitly aware that they are negotiating through a mediator agent, as the mediator agent interacts with the negotiation parties using the same protocol, in a proxy-like manner. The negotiation parties negotiate with the mediator agent as they would with a "regular" negotiation partner. This gives the mediator agent a flexible position in the negotiation process, as it can influence negotiations on both sides of the negotiation process. In this position, the mediator can pursue its own negotiation goals, which can for example be used to improve overall negotiation performance, by regulating the number of service providers involved in negotiations.

Application domains may differ with respect to the negotiation models/features they require. The negotiation framework is used to instantiate different negotiation models for different domains: For each particular domain, negotiation policies can be specified which govern the negotiation decisions made by the participating agents (template management, agreement request management, and agreement offer management decisions), aimed at creating a suitable negotiation model for each of the domains. In this thesis, the framework is applied in two different negotiation domains: Distributed energy management, and resource management in distributed agent middleware.

In the domain of distributed energy management, an important requirement with respect to the negotiation framework is the ability of the framework to create competition between the parties involved in the negotiation process. A number of negotiation models are instantiated using the negotiation framework, which result in different forms of competitive behavior: energy provider competition, competition between groups of energy providers, and competition between energy consumers.

The negotiation framework is also applied to the domain of distributed agent middleware. An important requirement for the negotiation framework in this domain is the ability for the framework to incorporate resource management policies defined by system administrators at both organizational and individual host levels. The negotiation framework is implemented as part of the AgentScape middleware, and allows agents to negotiate access to resources offered by (and through) the AgentScape platform on which they are running. Middleware resources (CPU-time, agent run-time environments (Java, C/C++, etc), and web service access) are the topics of negotiation. Before migrating to another location, agents in AgentScape first negotiate resource access with the location managers of potentially interesting locations: Agents request advertisements, and negotiate with selected locations to determine which location offers the best access conditions (i.e. the most CPU-time, access to specific web services, etc.). Location managers implement negotiation policies aimed at regulating resource usage throughout their locations, to achieve

management goals such as load balancing or fair-use.

The implementation of the framework in the AgentScape middleware contributes to the functionality needed to deploy AgentScape in environments that require regulation of resource usage, and that are managed de-centrally. This makes AgentScape one of the first agent infrastructures to include a negotiation based service management framework at the middleware level.

6.1 Research Questions

The introduction chapter presents four research questions. Each of these questions is revisited in this section, and the solutions provided by the framework are discussed.

Question 1: Can the negotiation framework support negotiation interactions over a wide range of service domains?
The WS-Agreement specification, upon which the negotiation language in the framework is based, provides basic structures for specifying advertisements, requests, offers, and final agreements. The specification is domain-independent, and defines placeholders ('service description terms' and 'negotiation constraints') which can contain domain-specific descriptions of services and constraint models. No assumptions are made about the format or content of these service descriptions. In our framework, a basic constraint model is defined allowing for the definition of generic minimum/maximum/enumeration constraints, applicable in many domains. The two use cases described in this thesis show how the framework can be instantiated in different domains.

Question 2: Can the negotiation framework provide a uniform architecture, suitable for application in these different service domains?
The framework specifies a mediated negotiation architecture, intended for domains in which *organizational structures* are part of the environment in which services are provided, and need to be explicitly incorporated in the service negotiation process. Mediators take on the role of representatives of these organizations. In most domains, organizations in one form or another can be distinguished, providing a straightforward mapping of the mediated negotiation architecture onto these organizations. Furthermore, the interaction protocol used in both tiers of the negotiation architecture consists of straightforward negotiation interactions (i.e. the exchange of negotiation documents), that can be easily implemented using different communication infrastructures (i.e. representing the documents as messages in a message-based communication infrastructure, or explicitly using interface calls as were specified in the framework description).

Question 3: Can the negotiation framework incorporate both organizational and local service access policies in the negotiation process?
The mediator agent of the negotiation framework is explicitly modeled as an active component within the negotiation process: The mediator agent gathers information on available services, and allows consumer agents to negotiate for these services. The mediator agent guides the negotiation process as it sees fit, by selectively choosing which parties

will be involved in the negotiation process. In our framework, mediator agents represent virtual provider organizations. Combining this role with the mediator role makes it possible to implement organizational policies concerning service usage by influencing the ongoing negotiation processes: The two primary ways in which the negotiation process is influenced by the mediator agent is through the advertisement combination process, and the service provider agent selection process. Local service access policies are enforced by the service provider agents in the negotiation framework, as they are responsible for negotiating with mediators on behalf of the service providers.

Question 4: Can the negotiation framework accommodate for dynamic and heterogeneous agent and provider populations?
Service access requirements of agents are subject to change. Different agents have different requirements, and requirements of agents change over time. Agreement negotiations are initiated by agents, based on these requirements. When requirements of agents change, and established agreement(s) are no longer sufficient, agents negotiate new agreements, reflecting the new requirement conditions. Old agreements expire, or are removed when new agreements are negotiated by agents, depending on the policies of the service providers. At the service provider agent side, changes in the provider population can either be the result of changes in the configuration of the provider organization (providers arrive or leave an organization), or changes in the configuration of an individual provider (services are added or removed, policies change). Advertisements are used to reflect these changes in the negotiation process: Provider agents periodically update their advertisements to reflect new conditions. The mediator agent, in turn, monitors the provider agent population, and periodically collects advertisements from the provider agents, resulting in an updated view of the organization.

6.2 Future Work

The research presented in this thesis can be extended in two areas: Extending the negotiation framework, and extending the implementation of the framework in the AgentScape agent middleware.

Extensions to the Framework

In many components of our negotiation framework, domain-specific policies govern the negotiation process. A language for specifying policies has not been proposed in this thesis. The WS-policy framework [10] includes a reasonably high-level policy specification language, which could be used in the framework as a basis for specifying negotiation policies at both the mediator and service provider agent levels.

Currently, the framework assumes that agents and service providers use a common language describing the domain-specific services. This assumption can be dropped if the mediator is also given the role of translator: This would allow agents and service providers to use different languages when negotiating with each other, provided that the mediator is capable of translating between the languages.

Our framework distinguishes two negotiation layers. More layers could be added to the model, if more than one organizational layer is needed: Service consumer agents could be grouped into virtual organizations just like service provider agents, allowing consumer agents to combine their individual demands, strengthening their position in the negotiation process. Clustering mechanisms which allow agents to autonomously form clusters based on their service requirements could be provided by the framework for this purpose (see Ogston et al. [71]). An alternative approach would be for mediator agents to group consumer negotiation requests, on behalf of consumers. However, in service oriented environments, consumer populations are more dynamic than service provider populations, making grouping of requests more complex.

Furthermore, the mediator in the framework can be given a more pro-active role in establishing agreements: The mediator could monitor negotiations, and based on this information, create frequently requested agreements in advance, and give requesting agents the option of accepting these 'default' agreements. This would reduce negotiation time from the consumer agent point-of-view, and give the mediator more control over the contents of negotiated agreements.

Extensions to the AgentScape Implementation

Additional resources can be modeled, allowing agents to negotiate for 'services' such as *communication bandwidth*, *disk space*, etc. This also requires that the AgentScape implementation supports monitoring and control of these resources, to enforce the agreements that are established.

Also, currently AgentScape administrators modify the negotiation advertisement information directly, to reflect service access conditions. A more user-friendly approach would be to define a policy language in which administrators can specify access conditions, which would be translated into advertisement information by the framework.

Finally, the implementation can be extended to allow agents to not only act as consumers, but also as providers of services. This would allow agent applications to expose 'services' to other agents, and negotiate for 'terms of use' of those services, for example acting as resource mediators for "dumb" Grid applications: In Grid system workflow management, parallel jobs have to be scheduled to the available Grid resources, during the different phases of the computation. Negotiation agents could be associated with such tasks, to manage the resource allocation for these Grid jobs by establishing agreements for the required resources.

Samenvatting

Agent-Gebaseerd Onderhandelen Over Services Via Mediators

In de komende jaren zullen grootschalige computernetwerken zoals het Internet in groeiende mate een basis vormen voor geautomatiseerde, autonome systemen, welke in staat zullen zijn om zonder tussenkomst van mensen taken uit te voeren in uitéénlopende gebieden. Deze systemen zullen bestaan uit grote aantallen individuele componenten (*Services*) die specifieke functionaliteiten aanbieden en dynamisch gecombineerd kunnen worden om complexere functionaliteit aan te bieden, al naar gelang de vraag. Het beheersen van deze grootschalige service georienteerde systemen (bijvoorbeeld voor het voorkomen/oplossen van falende componenten en infrastructuur, en het optimaliseren van de systeemconfiguratie) is één van de grote uitdagingen voor het *Service Oriented Computing* onderzoeksveld voor de komende jaren.

Software agenten zullen hierin een sleutelrol gaan vervullen. Agent-gebaseerde systemen worden gekarakteriseerd door eigenschappen zoals decentrale controle en autonomie van individuele agenten, en zijn daarom zeer geschikt voor het implementeren van gedistribueerde en autonome oplossingen in service georiënteerde dynamische omgevingen. Huidige agent platformen zijn echter nog niet geschikt om agenten in deze omgevingen te ondersteunen, aangezien er hoge eisen worden gesteld op gebieden zoals schaalbaarheid, veiligheid, openheid en beheersbaarheid.

Dit proefschrift richt zich op het laatstgenoemde probleem van beheersbaarheid: Om goed te kunnen functioneren moeten agenten niet alleen met elkaar kunnen communiceren, ze moeten ook toegang hebben tot benodigde *services* om specifieke taken te kunnen uitvoeren. In een grootschalige omgeving met vele agenten zijn deze services schaars, en gedistribueerd over meerdere administratieve domeinen (en dus onderhevig aan verschillend beleid met betrekking tot het beheer van deze services): Toegang tot de services moet gereguleerd kunnen worden voor redenen zoals veiligheid, het garanderen van systeemprestaties, en voor het uitvoeren van domein-specifieke beleidsbeslissingen.

Dit proefschrift stelt een oplossing voor gebaseerd op een onderhandelingsmodel, dat het mogelijk maakt om overeenkomsten te sluiten tussen aanbieders en afnemers van services. Onderhandelingsmodellen beschrijven interacties tussen deelnemers die het mogelijk maken om eisen/wensen uit te wisselen, waarbij uiteindelijk concensus wordt bereikt tussen de deelnemers over de uitkomst van een onderhandeling. Gedurende het onderhan-

delingsproces blijft de autonomie van de individuele deelnemers gerespecteerd.

Huidige oplossingen voor het onderhandelen over toegang tot services zijn veelal domein-specifiek, en bieden niet de praktische basis die nodig is om een breed scala aan onderhandelingsmodellen te kunnen implementeren. Standaarden zijn belangrijk voor adoptie en interoperabiliteit, en een aantal hiervan zijn in ontwikkeling, maar bieden nog niet de benodigde flexibiliteit om een goede basis te vormen.

In dit proefschrift wordt een agent-gebaseerd raamwerk voor onderhandelingen over services gepresenteerd, om onderhandelingen tussen aanbieders en afnemers van services vorm te kunnen geven. De gebruikte onderhandelingstaal en het onderhandelingsprotocol zijn gebaseerd op de Web Service Agreement (WS-Agreement) specificatie. Deze specificatie biedt een onderhandelingstaal en -protocol, waarin onderhandelingen gebaseerd zijn op zogenaamde *template* documenten, en de onderhandelingspartijen via een simpel *offer-agreement* protocol een overeenkomst kunnen sluiten. Het resultaat van onderhandelingen is een document waarin staat gespecificeerd welke services geconsumeerd mogen worden door de afnemer, en onder welke condities.

Om het onderhandelingsraamwerk de benodigde flexibiliteit te geven is in dit proefschrift het WS-Agreement protocol uitgebreid met een expliciete *acceptatie* fase, en is een zogenaamde *mediator* in het model opgenomen. Onderhandelingen vinden plaats op twee niveau's: Tussen afnemers en mediators enerzijds, en tussen de mediators en aanbieders anderszijds. De aanwezigheid van een mediator in het raamwerk maakt het mogelijk om verschillende onderhandelingsvormen te kunnen modelleren, waarbij de mediator de onderhandelingen reguleert.

In het proefschrift wordt het raamwerk beschreven en wordt een implementatie van het raamwerk beschreven in de context van de gedistribueerde multi-agent infrastructuur AgentScape. In AgentScape kunnen agenten zich verplaatsen tussen verschillende beschikbare lokaties (zogenaamde agent migratie). Agenten kunnen in AgentScape via de onderhandelingsinfrastructuur onderhandelen over toegang tot services (CPU, geheugen, database toegang, etc.), voordat tot migratie naar een AgentScape lokatie wordt overgegaan. Dit onderhandelen voor migratie heeft als groot voordeel dat voordat agenten daadwerkelijk gebruik gaan maken van de services van een lokatie, afspraken zijn gemaakt over de specifieke condities waaronder de toegang tot de services wordt verleend. Dit stelt beheerders van AgentScape lokaties en de onderliggende infrastructuur in staat om controle uit te oefenen en beleid uit te voeren aangaande het gebruik van de infrastructuur. Experimenten worden beschreven waarin het gebruik van het onderhandelingsraamwerk in AgentScape wordt geanalyseerd.

Om de flexibiliteit van het raamwerk verder te demonstreren worden een aantal onderhandelingsscenario's beschreven binnen het domein van *distributed energy management*. In dit domein is een groeiende behoefte aan geautomatiseerde onderhandelingssystemen om in te kunnen spelen op vrije en gedistribueerde energiemarkten. De scenario's beschrijven hoe het basisraamwerk kan worden gebruikt om onderhandelingen mogelijk te maken tussen aanbieders en afnemers van energie. Vervolgens wordt het raamwerk gebruikt om een veilingmodel vorm te geven, en om het mogelijk te maken voor deelnemers om onderhandelingen af te breken (*decommitment*). De scenario's worden ondersteund door simulaties die met behulp van de in AgentScape geïmplementeerde infrastructuur

zijn uitgevoerd.

Het gepresenteerde raamwerk biedt meer flexibiliteit dan de WS-Agreement specificatie waarop het is gebaseerd, onder andere door de uitbreiding van het protocol met een additionele fase en de expliciete rol van de mediator, die onderhandelingsprocessen kan sturen om zo verschillende onderhandelingsmodellen vorm te geven. Het raamwerk kan in de basisvorm succesvol ingezet worden in verschillende domeinen, en biedt voldoende ruimte voor uitbreiding om diverse onderhandelingsmodellen te ondersteunen. De implementatie van het raamwerk in AgentScape geeft het agent-platform de benodigde controle over toegang tot services door agenten die gebruik maken van het gedistribueerde platform. Deze controle is een belangrijke voorwaarde voor de acceptatie van AgentScape als ondersteuning van grootschalige, gedistribueerde systemen.

Bibliography

[1] Simple Object Access Protocol (SOAP). See http://www.w3.org/2000/xp/Group/.

[2] Distributed Generation in Liberalised Electricity Markets. International Energy Agentcy (IEA), 2002.

[3] New ERA for electricity in Europe – Distributed Generation: Key Issues, Challenges and Proposed Solutions. European Commission, Directorate-General for Research, 2003.

[4] T. Ackermann, G. Andersson, and Lennart Söder. Distributed generation: a definition. *Electric Power Systems Research*, 57:195–204, 2001.

[5] M. Aiello, G. Frankova, and D. Malfatti. What's in an Agreement? An Analysis and an Extension of WS-Agreement. In B. Benatallah, F. Casati, and P. Traverso, editors, *Proceedings of the Third International Conference on Service Oriented Computing (ICSOC)*, volume 3826 of *"Lecture Notes in Computer Science"*, pages 424–436. Springer, Amsterdam, The Netherlands, November 2005.

[6] H. Akkermans, F. Ygge, and R. Gustavsson. HOMEBOTS: Intelligent Decentralized Services for Energy Management. In *Proceedings of the Fourth International Symposium on the Management of Industrial and Corporate Knowledge (ISMICK'96)*, 1996.

[7] M. Albers, C.M. Jonker, M. Karami, and J. Treur. Agent Models and Different User Ontologies for an Electronic Market Place. *Knowledge and Information Systems*, 6(1):1–41, 2004.

[8] G. Allen, K. Davis, T. Goodale, A. Hutanu, H. Kaiser, T. Kielmann, A. Merzky, R. van Nieuwpoort, A. Reinefeld, F. Schintke, T. Schütt, E. Seidel, and B. Ullmer. The Grid Application Toolkit: Towards Generic and Easy Application Programming Interfaces for the Grid. *Proceedings of the IEEE*, 93(3):534–550, 2005.

[9] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specification WS-Agreement (draft). Global Grid Forum, 2004.

[10] S. Bajaj, D. Box, D. Chappel, F. Curbera, G. Daniels, P. Hallam-Baker, M. Hondo, C. Kaler, D. Langworthy, A. Nadalin, N. Nagaratnam, H. Prafullchandra, C. von Riegen, D. Roth, J. Schlimmer, C. Sharp, J. Shewchuk, A. Vedamuthu, . Yalnalp, and D. Orchard. Web Services Policy Framework (WSPolicy), 2006.

[11] C. Baumer, M. Breugst, S. Choy, and T. Magedanz. Grasshopper: a universal agent platform based on OMG MASIF and FIPA standards. Technical report, IKV++ GmbH, 2000.

[12] R. A. Belecheanu, S. Munroe, M. Luck, T. Payne, T. Miller, P. McBurney, and M. Pechoucek. Commercial Applications of Agents: Lessons, Experiences and Challenges. In *Proceedings of the Fifth International Conference on Autonomous Agents and Multiagent Systems*, Hakodate, Japan, 2006.

[13] F. Bellifemine, A. Poggi, and G. Rimassa. Developing Multi-agent Systems with JADE. In C. Castelfranchi and Y. Lesprance, editor, *Intelligent Agents VII. Agent Theories Architectures and Languages (ATAL 2000)*, volume 1986 / 2001 of *Lecture Notes in Computer Science*. Springer, Berlin, Germany, 2001.

[14] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, and J. Siméon. XML Path Language (XPath) 2.0, 1999. http://www.w3.org/TR/xpath20/.

[15] M. Bichler, G. Kersten, and S. Strecker. Towards a Structured Design of Electronic Negotiations. *Group Decision and Negotiation"*, 12(4):311–335, 2003.

[16] W. Binder, J. G. Hulaas, and A. Villazon. Portable Resource Control in Java. In *Proceedings of the 16th ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications*, pages 139–155. ACM Press, 2001.

[17] C. Bohoris, G. Pavlou, and H. Cruickshank. Using Mobile Agents for Network Performance Management. In *Proceedings of the IFIP/IEEE Network Operations and Management Symposium (NOMS'00)*, Hawaii, USA, 2000.

[18] F.M.T. Brazier, F. Cornelissen, R. Gustavsson, C.M. Jonker, O. Lindeberg, B. Polak, and J. Treur. A Multi-Agent System Performing One-to-Many Negotiation for Load Balancing of Electricity Use. *Electronic Commerce Research and Applications Journal*, 1:208–224, 2002.

[19] F.M.T. Brazier, C.M. Jonker, and J. Treur. Principles of Component-Based Design of Intelligent Agents. *Data and Knowledge Engineering*, 41:1–28, 2002.

[20] R. B. Bunt, D. L. Eager, G. M. Oster, and C. L. Williamson. Achieving load balance and effective caching in clustered Web servers. In *Proceedings of the 4th International Web Caching Workshop*, 1999.

[21] R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid. In *Proceedings of HPC ASIA*, Beijing, China, 2000. IEEE CS Press.

[22] H.L. Cardoso and E. Oliveira. Assisting and Regulating Virtual Enterprise Interoperability through Contracts. In K. Fischer, A. Berre, K. Elms, and J.P. Muller, editors, *Proceedings of the AAMAS 2005 Workshop Agent-based Technologies and Applications for Enterprise Interoperability (ATOP)*, pages 1–12, Utrecht, The Netherlands, July 2005.

[23] H. Casanova and J. Dongarra. NetSolve: A Network-Enabled Server for Solving Computational Science Problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, 1997.

[24] A. Chavez and P. Maes. Kasbah: An Agent Marketplace for Buying and Selling Goods. In *First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96)*, pages 75–90, London, UK, 1996. Practical Application Company.

[25] Y. Chevaleyre, P. E. Dunne, U. Endriss, J. Lang, M. Lemaitre, N. Maudet, J. Padget, S. Phelps, J. A. Rodriguez-Aguilar, and P. Sousa. Issues in Multiagent Resource Allocation. *Informatica*, 30(1):3–31, 2006.

[26] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: an Overlay Testbed for Broad-coverage Services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, 2003.

[27] J. Collins, B. Youngdahl, S. Jamison, B. Mobasher, and M. Gini. A Market Architecture for Multi-Agent Contracting. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 285–292, New York, 9–13, 1998. ACM Press.

[28] P. Cramton. Simultaneous Ascending Auction. Papers of Peter Cramton 04mit4, University of Maryland, Department of Economics - Peter Cramton, 2004. available at http://ideas.repec.org/p/pcc/pccumd/04mit4.html.

[29] P. Cramton, J. McMillan, P. Milgrom, B. Miller, B. Mitchell, and D. Vincent. Simultaneous Ascending Auctions with Package Bidding. Papers of Peter Cramton 98cra2, University of Maryland, Department of Economics - Peter Cramton, March 1998. available at http://ideas.repec.org/p/pcc/pccumd/98cra2.html.

[30] V. Darley and D. Sanders. An Agent-Based Model of a Corrugated-Box Factory: The Trade-Off Between Finished Goods Stock and On-Time-In-Full Delivery. In H. Coelho and B. Espinasse, editors, *Proceedings of the Fifth Workshop on Agent-Based Simulation*, 2004.

[31] V. Darley and D. Sanders. An Agent-based Model of a Corrugated-box Factory: The Trade-off between Finished Goods Stock and On-time-in-full Delivery. In H. Coelho and B. Espinasse, editors, *Proceedings of the Fifth Workshop on Agent-Based Simulation*, 2004.

[32] E. David, R. Azoulay-Schwartz, and S. Kraus. Protocols and strategies for automated multi-attribute auctions. In *AAMAS '02: Proceedings of the first international*

joint conference on Autonomous agents and multiagent systems, pages 77–85, New York, NY, USA, 2002. ACM Press.

[33] D. Deugo, M. Weiss, and E. Kendall. *Reusable patterns for agent coordination*, pages 347–368. Springer-Verlag, London, UK, 2001.

[34] V. Dignum, J-J. Meyer, F. Dignum, and H. Weigand. Formal Specification of Interaction in Agent Societies. In "M. Hinchey, J. Rash, W. Truszkowski, C. Rouff, and D. Gordon-Spears", editors, *Formal Approaches to Agent-Based Systems (FAABS)*, "Lecture Notes in Computer Science". Springer, Berlin, Germany, 2003.

[35] Virginia Dignum and Frank Dignum. Modelling Agent Societies: Co-ordination Frameworks and Institutions. In *Portuguese Conference on Artificial Intelligence*, pages 191–204, 2001.

[36] K. Dorer and M. Calisti. An Adaptive Solution to Dynamic Transport Optimization. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 45–51, New York, NY, USA, 2005. ACM Press.

[37] C. B. Excelente-Toledo, R. A. Bourne, and N. R. Jennings. Reasoning about Commitments and Penalties for Coordination between Autonomous Agents. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 131–138, Montreal, Canada, 2001. ACM Press.

[38] P. Faratin, C. Sierra, N. R. Jennings, and P. Buckle. Designing Responsive and Deliberative Automated Negotiators. In *Proceedings of the AAAI Workshop on Negotiation: Settling Conflicts and Identifying Opportunities*, pages 12–18, Orlando, FL, 1999.

[39] G. Federico and D. Rahman. Bidding in an electricity pay-as-bid auction. Economics Papers 2001-W5, Economics Group, Nuffield College, University of Oxford, April 2000. available at http://ideas.repec.org/p/nuf/econwp/0105.html.

[40] I. Foster, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, H. Kishimoto, F. Maciel, A. Savva, F. Siebenlist, R. Subramaniam, J. Treadwell, , and J.V. Reich. The Open Grid Services Architecture, 2004.

[41] I. Foster, N. Jennings, and C. Kesselman. Brain Meets Brawn: Why Grid and Agents Need Each Other. *Autonomous Agents and Multi-Agent Systems*, 2004.

[42] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kauffman, 1998.

[43] D. Gannon, B. Plale, M. Christie, L. Fang, Y. Huang, S. Jensen, G. Kandaswamy, S. Marru, S. L. Pallickara, S. Shirasuna, Y. Simmhan, A. Slominski, and Y. Sun. Service Oriented Architectures for Science Gateways on Grid Systems. In B. Benatallah, F. Casati, and P. Traverso, editors, *Proceedings of the Third International Conference on Service Oriented Computing (ICSOC)*, volume 3826 of *"Lecture Notes in*

Computer Science", pages 21–32. Springer, Amsterdam, The Netherlands, November 2005.

[44] E. Gerding, D. van Bragt, and J. Poutre. Scientific Approaches and Techniques for Negotiation: a Game Theoretic and Artificial Intelligence Perspective. Technical Report SEN-R0005, CWI, Amsterdam, The Netherlands, 2000.

[45] A. Helsinger, M. Thome, and T. Wright. Cougaar: A scalable, distributed multi-agent archi- tecture. In *Proceedings of the International Conference on Systems, Man and Cybernetics (IEEE SMC 2004)*, The Hague, The Netherlands, October 2004.

[46] K. V. Hindriks, F. S. de Boer, W. ven der Hoek, and J.-J. Ch. Meyer. Semantics of communicating agents based on deduction and abduction. In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*, pages 63–79. Springer-Verlag: Heidelberg, Germany, 2000.

[47] P. C. K. Hung, H. Li, and J. Jeng. WS-Negotiation: An Overview of Research Issues. In *Proceedings of the 37th Hawaii International Conference on System Sciences (HICSS'04)*, pages 33–42, Big Island, Hawaii, January 2004.

[48] G. James, D. Cohen, R. Dodier, G. Platt, and D. Palmer. A deployed multi-agent framework for distributed energy applications. In *5th International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS 2006),Hakodate, Japan*, May 2006.

[49] N. Jennings, S. Parsons, C. Sierra, and P. Faratin. Automated Negotiation. In *Proceedings of the 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Systems*, pages 23–30, Manchester, UK, 2000.

[50] N. R. Jennings, K. Sycara, and M. Wooldridge. A Roadmap of Agent Research and Development. *Journal of Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[51] N. Karnik and A. Tripathi. Agent Server Architecture for the Ajanta Mobile-Agent System. In *Proceedings of the 1998 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, Las Vegas, NV, USA, 1998.

[52] A. Keller and H. Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management*, 11(1):57–81, March 2003.

[53] P. Klemperer. Auction Theory: A Guide to the Literature. Microeconomics 9903002, EconWPA, March 1999. available at http://ideas.repec.org/p/wpa/wuwpmi/9903002.html.

[54] J. K. Kok, C. J. Warmer, and I. G. Kamphuis. PowerMatcher: Multiagent Control in the Electricity Infrastructure. In *AAMAS '05: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 75–82, New York, NY, USA, 2005. ACM Press.

[55] S. Kraus, J. Wilkenfeld, and G. Zlotkin. Multiagent Negotiation under Time Constraints. *Artificial Intelligence*, 75(2):297–345, 1995.

[56] K. Krauter, R. Buyya, and M. Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software Practice and Experience*, 32(2):135–164, 2002.

[57] B. Krulwich. The BargainFinder Agent: Comparison Price Shopping on the Internet. In *Agents, Bots, and other Internet Beasties*. Macmillan Publishing, 1996.

[58] G. Lai, C. Li, K. Sycara, and J. A. Giampapa. Literature Review on Multi-attribute Negotiations. Technical Report CMU-RI-TR-04-66, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 2004.

[59] D. B. Lange and M. Oshima. Seven Good Reasons for Mobile Agents. *Commun. ACM*, 42(3):88–89, 1999.

[60] D. B. Lange, M. Oshima, G. Karjoth, and K. Kosaka. Aglets: Programming mobile agents in Java. In T. Masuda, Y. Masunaga, and M. Tsukamoto, editors, *Worldwide Computing and Its Applications*, volume 1274 of *Lecture Notes in Computer Science*, pages 253–266. Springer Berlin / Heidelberg, 1997.

[61] F. Leymann. The (Service) Bus: Services Penetrate Everyday Life. In B. Benatallah, F. Casati, and P. Traverso, editors, *Proceedings of the Third International Conference on Service Oriented Computing (ICSOC)*, volume 3826 of *"Lecture Notes in Computer Science"*, pages 12–20. Springer, Amsterdam, The Netherlands, November 2005.

[62] M. Luck, P. McBurney, O. Shehory, and S. Willmott. *Agent Technology: Computing as Interaction (A Roadmap for Agent-Based Computing)*. AgentLink, 2005.

[63] H. Ludwig, A. Dan, and R. Keaney. Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements. Technical report, IBM Research Division, June 2004.

[64] O. Marin, M. Bertier, and P. Sens. DARX - A Framework for the Fault-Tolerant Support of Agent Software. In *Proceedings of the 14th. IEEE International Symposium on Software Reliability Engineering (ISSRE 2003)*, pages 406–417, November 2003.

[65] P. Mathieu and M. Verrons. A Generic Model for Contract Negotiation. In *Proceedings of the AISB'02 Convention*, pages 1–8, London, UK, April 2002.

[66] P. Milgrom. Putting Auction Theory to Work: The Simultaneous Ascending Auction. *Journal of Political Economy*, 108(2):245–272, April 2000. available at http://ideas.repec.org/a/ucp/jpolec/v108y2000i2p245-272.html.

[67] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations". Technical report, Santa Fe Institute, Santa Fe, 1996.

[68] T. J. Norman, A. Preece, S. Chalmers, N. R. Jennings, M. Luck, V. D. Dang, T. D. Nguyen, V. Deora, J. Shao, W. A. Gray, and N. J. Fiddian. Conoise: Agent-based formation of virtual organisations. *Research and Development in Intelligent SystemsXX: Proceedings of AI2003, the Twentythird SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 353–366, 2003.

[69] H. S Nwana, D. T. Ndumu, L. C. Lee, and J. C. Collis. ZEUS: a toolkit and approach for building distributed multi-agent systems. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 360–361, Seattle, WA, USA, 1999. ACM Press.

[70] J. Odell, H. Parunak, and B. Bauer. Extending UML for Agents. In *Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence*, 2000.

[71] E. Ogston, M. van Steen, and F.M.T. Brazier. Group formation among decentralized autonomous agents. *Applied Artificial Intelligence*, pages 953–970, October-December 2004.

[72] A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, editors. *Coordination of Internet Agents: Models, Technologies, and Applications*. Springer, 2001.

[73] B.J. Overeinder, F.M.T. Brazier, and O. Marin. Fault-Tolerance in Scalable Agent Support Systems: Integrating DARX in the AgentScape Framework. In *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2003)*, pages 688–695, May 2003.

[74] S. Paurobally and N. R. Jennings. Developing Agent Web Service Agreements. In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 464–470, Compiegne, France, September 2005.

[75] S. Paurobally and N.R. Jennings. Developing Agent Web Service Agreements. In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, 2005.

[76] A. Puliafito and O. Tomarchio. Using Mobile Agents to implement flexible Network Management strategies. *Computer Communication Journal*, 23(8):708–719, April 2000.

[77] H. Raiffa. *The Art and Science of Negotiation*. Cambridge, MA: The Belknap Press of Harvard University Press, 1982, 1982.

[78] R. Raman. *Matchmaking Frameworks for Distributed Resource Management*. PhD thesis, University of Wisconsin, 2000.

[79] A. S. Rao. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In R. van Hoe, editor, *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Eindhoven, The Netherlands, 1996.

[80] S. J. Rassenti, V. L. Smith, and B. J. Wilson. Discriminatory Price Auctions in Electricity Markets: Low Volatility at the Expense of High Price Levels. *Journal of Regulatory Economics*, 23(2):109–23, March 2003. available at http://ideas.repec.org/a/kap/regeco/v23y2003i2p109-23.html.

[81] D. M. Reeves, M. P. Wellman, and B. N. Grosof. Automated Negotiation from Declarative Contract Descriptions. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 51–58, Montreal, Canada, 2001. ACM Press.

[82] A. Rubinstein. Perfect Equilibrium in a Bargaining Model. *Econometrica*, 50(1):97–109, January 1982.

[83] T. Sandholm. An Implementation of the Contract Net Protocol Based on Marginal Cost Calculations. In *Proceedings of the National Conference on Artificial Intelligence*. AAAI Press, 1993.

[84] T. Sandholm and V. Lesser. Issues in Automated Negotiation and Electronic Commerce: Extending the Contract Net Framework. In Victor Lesser, editor, *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*, pages 328–335, San Francisco, CA, USA, 1995. The MIT Press: Cambridge, MA, USA.

[85] T. Sandholm and V. Lesser. Issues in Automated Negotiation and Electronic Commerce: Extending the Contract Net Framework. In Victor Lesser, editor, *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*, pages 328–335, San Francisco, CA, USA, 1995. The MIT Press: Cambridge, MA, USA.

[86] T. Sandholm, S. Sikka, and S. Norden. Algorithms for Optimizing Leveled Commitment Contracts. In *IJCAI*, pages 535–541, 1999.

[87] R. G. Smith. *The Contract Net Protocol: High-level Communication and Control in a Distributed Problem Solver*, pages 357–366. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[88] M. Ströbel and C. Weinhardt. The Montreal Taxonomy for Electronic Negotiations. *Group Decision and Negotiation*, 12:143–164, March 2003.

[89] N. Suri, J. Bradshaw, M. R. Breedy, P. T. Groth, G. A. Hill, and R. Jeffers. Strong Mobility and Fine-Grained Resource Control in NOMADS. In *ASA/MA 2000: Proceedings of the Second International Symposium on Agent Systems and Applications and Fourth International Symposium on Mobile Agents*, pages 2–15, London, UK, 2000. Springer-Verlag.

[90] Y. Tan and W. Thoen. DocLog: An Electronic Contract Representation Language. In *Proceedings of the DEXA 2000 Workshop*, 2000.

[91] R.J. Timmer. An efficient implementation of the agent operating system. Master's thesis, Vrije Universiteit Amsterdam, August 2005.

[92] C. Triki, P. Beraldi, and G. Gross. Optimal capacity allocation in multi-auction electricity markets under uncertainty. *Comput. Oper. Res.*, 32(2):201–217, 2005.

[93] G. van 't Noordende, F.M.T. Brazier, and A.S. Tanenbaum. Security in a Mobile Agent System. In *Proceedings of the First IEEE Symposium on Multi-Agent Security and Survivability*, Philadelphia, August 2004.

[94] D. Veit, W. Fichtner, and M. Ragwitz. Agent-based Computational Economics in Power Markets Multi-agent Based Simulation as a Tool for Decision Support. In J. Andrysek, M. Karny, and J. Kracik, editors, *Multiple Participant Decision Making*, volume 9 of *International Series on Advanced Intelligence*. Advanced Knowledge International, Adelaide, Australia, 2004.

[95] H. Weigand, A. de Moor, M. Schoop, and F. Dignum. B2B Negotiation Support: The Need for a Communication Perspective. *Group Decision and Negotiation*, 12:3–29, 2003.

[96] N.J.E. Wijngaards, B.J. Overeinder, M. van Steen, and F.M.T. Brazier. Supporting internet-scale multi-agent systems. *Data and Knowledge Engineering*, 41(2-3):229–245, June 2002.

[97] S. Woods and M. Barbacci. Architectural evaluation of collaborative agent-based systems. Technical report, Software Engineering Institute, Carnegie Mellon University, 1999.

[98] M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[99] R.E. Wray and R.M Jones. An Introduction to Soar as an Agent Architecture. In R. Sun, editor, *Cognition and Multi-agent Interaction: From Cognitive Modeling to Social Simulation*, pages 53–78. Cambridge University Press, 2005.

SIKS Dissertation Series

1998

1998-1 Johan van den Akker (CWI)

DEGAS - An Active, Temporal Database of Autonomous Objects

1998-2 Floris Wiesman (UM)

Information Retrieval by Graphically Browsing Meta-Information

1998-3 Ans Steuten (TUD)

A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective

1998-4 Dennis Breuker (UM)

Memory versus Search in Games

1998-5 E.W.Oskamp (RUL)

Computerondersteuning bij Straftoemeting

1999

1999-1 Mark Sloof (VU)

Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products

1999-2 Rob Potharst (EUR)

Classification using decision trees and neural nets

1999-3 Don Beal (UM)

The Nature of Minimax Search

1999-4 Jacques Penders (UM)

The practical Art of Moving Physical Objects

1999-5 Aldo de Moor (KUB)

Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems

1999-6 Niek J.E. Wijngaards (VU)

Re-design of compositional systems

1999-7 David Spelt (UT)

Verification support for object database design

1999-8 Jacques H.J. Lenting (UM)

Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation

2000

2000-1 Frank Niessink (VU)

Perspectives on Improving Software Maintenance

2000-2 Koen Holtman (TUE)

Prototyping of CMS Storage Management

2000-3 Carolien M.T. Metselaar (UVA)

Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief

2000-4 Geert de Haan (VU)

ETAG, A Formal Model of Competence Knowledge for User Interface Design

2000-5 Ruud van der Pol (UM)

Knowledge-based Query Formulation in Information Retrieval

2000-6 Rogier van Eijk (UU)

Programming Languages for Agent Communication

2000-7 Niels Peek (UU)

Decision-theoretic Planning of Clinical Patient Management

2000-8 Veerle Coup (EUR)

Sensitivity Analyis of Decision-Theoretic Networks

2000-9 Florian Waas (CWI)

Principles of Probabilistic Query Optimization

2000-10 Niels Nes (CWI)

Image Database Management System Design Considerations, Algorithms and Architecture

2000-11 Jonas Karlsson (CWI)

Scalable Distributed Data Structures for Database Management

2001

2001-1 Silja Renooij (UU)

Qualitative Approaches to Quantifying Probabilistic Networks

2001-2 Koen Hindriks (UU)

Agent Programming Languages: Programming with Mental Models

2001-3 Maarten van Someren (UvA)

Learning as problem solving

2001-4 Evgueni Smirnov (UM)

Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets

2001-5 Jacco van Ossenbruggen (VU)

Processing Structured Hypermedia: A Matter of Style

2001-6 Martijn van Welie (VU)

Task-based User Interface Design

2001-7 Bastiaan Schonhage (VU)

Diva: Architectural Perspectives on Information Visualization

2001-8 Pascal van Eck (VU)

A Compositional Semantic Structure for Multi-Agent Systems Dynamics

2001-9 Pieter Jan 't Hoen (RUL)

Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes

2001-10 Maarten Sierhuis (UvA)

Modeling and Simulating Work Practice BRAHMS: a multiagent modeling and simulation language for work practice analysis and design

2001-11 Tom M. van Engers (VUA)

Knowledge Management: The Role of Mental Models in Business Systems Design

2002

2002-01 Nico Lassing (VU)

Architecture-Level Modifiability Analysis

2002-02 Roelof van Zwol (UT)

Modelling and searching web-based document collections

2002-03 Henk Ernst Blok (UT)

Database Optimization Aspects for Information Retrieval

2002-04 Juan Roberto Castelo Valdueza (UU)

The Discrete Acyclic Digraph Markov Model in Data Mining

2002-05 Radu Serban (VU)

The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents

2002-06 Laurens Mommers (UL)

Applied legal epistemology; Building a knowledge-based ontology of the legal domain

2002-07 Peter Boncz (CWI)

Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications

2002-08 Jaap Gordijn (VU)

Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas

2002-09 Willem-Jan van den Heuvel(KUB)

Integrating Modern Business Applications with Objectified Legacy Systems

2002-10 Brian Sheppard (UM)

Towards Perfect Play of Scrabble

2002-11 Wouter C.A. Wijngaards (VU)

Agent Based Modelling of Dynamics: Biological and Organisational Applications

2002-12 Albrecht Schmidt (Uva)

Processing XML in Database Systems

2002-13 Hongjing Wu (TUE)

A Reference Architecture for Adaptive Hypermedia Applications

2002-14 Wieke de Vries (UU)

Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems

2002-15 Rik Eshuis (UT)

Semantics and Verification of UML Activity Diagrams for Workflow Modelling

2002-16 Pieter van Langen (VU)

The Anatomy of Design: Foundations, Models and Applications

2002-17 Stefan Manegold (UVA)

Understanding, Modeling, and Improving Main-Memory Database Performance

2003

2003-01 Heiner Stuckenschmidt (VU)

Ontology-Based Information Sharing in Weakly Structured Environments

2003-02 Jan Broersen (VU)

Modal Action Logics for Reasoning About Reactive Systems

2003-03 Martijn Schuemie (TUD)

Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy

2003-04 Milan Petkovic (UT)

Content-Based Video Retrieval Supported by Database Technology

2003-05 Jos Lehmann (UVA)

Causation in Artificial Intelligence and Law - A modelling approach

2003-06 Boris van Schooten (UT)

Development and specification of virtual environments

2003-07 Machiel Jansen (UvA)

Formal Explorations of Knowledge Intensive Tasks

2003-08 Yongping Ran (UM)

Repair Based Scheduling

2003-09 Rens Kortmann (UM)

The resolution of visually guided behaviour

2003-10 Andreas Lincke (UvT)

Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture

2003-11 Simon Keizer (UT)

Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks

2003-12 Roeland Ordelman (UT)

Dutch speech recognition in multimedia information retrieval

2003-13 Jeroen Donkers (UM)

Nosce Hostem - Searching with Opponent Models

2003-14 Stijn Hoppenbrouwers (KUN)

Freezing Language: Conceptualisation Processes across ICT-Supported Organisations

2003-15 Mathijs de Weerdt (TUD)

Plan Merging in Multi-Agent Systems

2003-16 Menzo Windhouwer (CWI)

Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses

2003-17 David Jansen (UT)

Extensions of Statecharts with Probability, Time, and Stochastic Timing

2003-18 Levente Kocsis (UM)

Learning Search Decisions

2004

2004-01 Virginia Dignum (UU)

A Model for Organizational Interaction: Based on Agents, Founded in Logic

2004-02 Lai Xu (UvT)

Monitoring Multi-party Contracts for E-business

2004-03 Perry Groot (VU)

A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving

2004-04 Chris van Aart (UVA)

Organizational Principles for Multi-Agent Architectures

2004-05 Viara Popova (EUR)

Knowledge discovery and monotonicity

2004-06 Bart-Jan Hommes (TUD)

The Evaluation of Business Process Modeling Techniques

2004-07 Elise Boltjes (UM)

Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes

2004-08 Joop Verbeek(UM)

Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiële gegevensuitwisseling en digitale expertise

2004-09 Martin Caminada (VU)

For the Sake of the Argument; explorations into argument-based reasoning

2004-10 Suzanne Kabel (UVA)

Knowledge-rich indexing of learning-objects

2004-11 Michel Klein (VU)

Change Management for Distributed Ontologies

2004-12 The Duy Bui (UT)

Creating emotions and facial expressions for embodied agents

2004-13 Wojciech Jamroga (UT)

Using Multiple Models of Reality: On Agents who Know how to Play

2004-14 Paul Harrenstein (UU)

Logic in Conflict. Logical Explorations in Strategic Equilibrium

2004-15 Arno Knobbe (UU)

Multi-Relational Data Mining

2004-16 Federico Divina (VU)

Hybrid Genetic Relational Search for Inductive Learning

2004-17 Mark Winands (UM)

Informed Search in Complex Games

2004-18 Vania Bessa Machado (UvA)

Supporting the Construction of Qualitative Knowledge Models

2004-19 Thijs Westerveld (UT)

Using generative probabilistic models for multimedia retrieval

2004-20 Madelon Evers (Nyenrode)

Learning from Design: facilitating multidisciplinary design teams

2005

2005-01 Floor Verdenius (UVA)

Methodological Aspects of Designing Induction-Based Applications

2005-02 Erik van der Werf (UM))

AI techniques for the game of Go

2005-03 Franc Grootjen (RUN)

A Pragmatic Approach to the Conceptualisation of Language

2005-04 Nirvana Meratnia (UT)

Towards Database Support for Moving Object data

2005-05 Gabriel Infante-Lopez (UVA)

Two-Level Probabilistic Grammars for Natural Language Parsing

2005-06 Pieter Spronck (UM)

Adaptive Game AI

2005-07 Flavius Frasincar (TUE)

Hypermedia Presentation Generation for Semantic Web Information Systems

2005-08 Richard Vdovjak (TUE)

A Model-driven Approach for Building Distributed Ontology-based Web Applications

2005-09 Jeen Broekstra (VU)

Storage, Querying and Inferencing for Semantic Web Languages

2005-10 Anders Bouwer (UVA)

Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments

2005-11 Elth Ogston (VU)

Agent Based Matchmaking and Clustering - A Decentralized Approach to Search

2005-12 Csaba Boer (EUR)

Distributed Simulation in Industry

2005-13 Fred Hamburg (UL)

Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen

2005-14 Borys Omelayenko (VU)

Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics

2005-15 Tibor Bosse (VU)

Analysis of the Dynamics of Cognitive Processes

2005-16 Joris Graaumans (UU)

Usability of XML Query Languages

2005-17 Boris Shishkov (TUD)

Software Specification Based on Re-usable Business Components

2005-18 Danielle Sent (UU)

Test-selection strategies for probabilistic networks

2005-19 Michel van Dartel (UM)

Situated Representation

2005-20 Cristina Coteanu (UL)

Cyber Consumer Law, State of the Art and Perspectives

2005-21 Wijnand Derks (UT)

Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics

2006

2006-01 Samuil Angelov (TUE)

Foundations of B2B Electronic Contracting

2006-02 Cristina Chisalita (VU)

Contextual issues in the design and use of information technology in organizations

2006-03 Noor Christoph (UVA)

The role of metacognitive skills in learning to solve problems

2006-04 Marta Sabou (VU)

Building Web Service Ontologies

2006-05 Cees Pierik (UU)

Validation Techniques for Object-Oriented Proof Outlines

2006-06 Ziv Baida (VU)

Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling

2006-07 Marko Smiljanic (UT)

XML schema matching – balancing efficiency and effectiveness by means of clustering

2006-08 Eelco Herder (UT)

Forward, Back and Home Again - Analyzing User Behavior on the Web

2006-09 Mohamed Wahdan (UM)

Automatic Formulation of the Auditor's Opinion

2006-10 Ronny Siebes (VU)

Semantic Routing in Peer-to-Peer Systems

2006-11 Joeri van Ruth (UT)

Flattening Queries over Nested Data Types

2006-12 Bert Bongers (VU)

Interactivation - Towards an e-cology of people, our technological environment, and the arts

2006-13 Henk-Jan Lebbink (UU)

Dialogue and Decision Games for Information Exchanging Agents

2006-14 Johan Hoorn (VU)

Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change

2006-15 Rainer Malik (UU)

CONAN: Text Mining in the Biomedical Domain

2006-16 Carsten Riggelsen (UU)

Approximation Methods for Efficient Learning of Bayesian Networks

2006-17 Stacey Nagata (UU)

User Assistance for Multitasking with Interruptions on a Mobile Device

2006-18 Valentin Zhizhkun (UVA)

Graph transformation for Natural Language Processing

2006-19 Birna van Riemsdijk (UU)

Cognitive Agent Programming: A Semantic Approach

2006-20 Marina Velikova (UvT)

Monotone models for prediction in data mining

2006-21 Bas van Gils (RUN)

Aptness on the Web

2006-22 Paul de Vrieze (RUN)

Fundaments of Adaptive Personalisation

2006-23 Ion Juvina (UU)

Development of Cognitive Model for Navigating on the Web

2006-24 Laura Hollink (VU)

Semantic Annotation for Retrieval of Visual Resources

2006-25 Madalina Drugan (UU)

Conditional log-likelihood MDL and Evolutionary MCMC

2006-26 Vojkan Mihajlovic (UT)

Score Region Algebra: A Flexible Framework for Structured Information Retrieval

2006-27 Stefano Bocconi (CWI)

Vox Populi: generating video documentaries from semantically annotated media repositories

2006-28 Borkur Sigurbjornsson (UVA)

Focused Information Access using XML Element Retrieval

2007

2007-01 Kees Leune (UvT)

Access Control and Service-Oriented Architectures

2007-02 Wouter Teepe (RUG)

Reconciling Information Exchange and Confidentiality: A Formal Approach

2007-03 Peter Mika (VU)

Social Networks and the Semantic Web

2007-04 Jurriaan van Diggelen (UU)

Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach

2007-05 Bart Schermer (UL)

Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance

2007-06 Gilad Mishne (UVA)

Applied Text Analytics for Blogs

2007-07 Natasa Jovanovic' (UT)

To Whom It May Concern - Addressee Identification in Face-to-Face Meetings

2007-08 Mark Hoogendoorn (VU)

Modeling of Change in Multi-Agent Organizations